ESD-TDR-65-68

Technical Report

387

On-Line Documentation of the Compatible Time-Sharing System

J. M. Winett

12 May 1965

Prepared under Electronic Systems Division Contract AF 19(628)-500 by

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

The work reported in this document was performed at Lincoln Laboratory, a center for research operated by Massachusetts Institute of Technology, with the support of the U.S. Air Force under Contract AF 19(628)-500. The computer time was supported by Project MAC, an M.I.T. research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LINCOLN LABORATORY

ON-LINE DOCUMENTATION OF THE COMPATIBLE TIME-SHARING SYSTEM

J. M. WINETT

Group 28

TECHNICAL REPORT 387

12 MAY 1965

LEXINGTON MASSACHUSETTS

ON-LINE DOCUMENTATION OF THE COMPATIBLE TIME-SHARING SYSTEM*

ABSTRACT

The dissemination of informotion about computer progroms is hompered because of the lack of conformity in documentotion, the delays inherent in any distribution system, and the inability to select only desired informotion without being flooded with informotion which is not of present interest. An on-line system for storing ond retrieving information about the programs associated with the Compatible Time-Shoring System (CTSS) has been developed to be included os o CTSS commond. This system will help to document the system commands, supervisor entries, librory subprogroms, ond public programs. These types of programs have been chosen since there is an urgent need for hoving this documentation ovoilable on demond, i.e., on-line.

Accepted for the Air Force
Stonley J. Wisniewski
Lt Colonel, USAF
Chief, Lincoln Loborotory Office

* This report is bosed on a thesis of the some title submitted to the Department of Electrical Engineering ot the Mossochusetts Institute of Technology on 18 Jonuory 1965, in portiol fulfillment of the requirements for the Degree of Electrical Engineer.

TABLE OF CONTENTS

ACKNOWLEDGMENT

The author acknowledges the suppart given by the M. I. T. Lincoln Laboratary under the Staff Associate Pragram far the pursuance af the graduate wark far which this thesis has been a part. The otmosphere far carrying out this reseorch and the camputer time on the Compatible Time-Sharing System has been provided by Praject MAC.

The author would like ta express his appreciatian to the members of the Mechanical Translation Group af the Research Laboratory af Electranics, under the supervision af Dr. Victor Yngve, far adopting COMIT for use with CTSS and for their cantinuing efforts ta make COMIT mare useful as a pragramming language. The authar is grateful to Professor Corbató for his supervision af this thesis, and wauld like ta give speciol recognition to him far the development of the Campatible Time-Sharing System.

ON-LINE DOCUMENTATION OF THE COMPATIBLE TIME-SHARING SYSTEM

I. INTRODUCTION

The dissemination of information about computer programs is hampered by the lack of conformity in documentation, the delays inherent in any distribution system, and the inability to select only desired information without being flooded with information which is not of present interest. An on-line system for storing and retrieving information about the programs associated with the Compatible Time-Sharing System (CTSS) has been developed to be included as a CTSS command (Ref. 1). This system will help to document the system commands, supervisor entries, library subprograms, and public programs. These categories of programs have been chosen because there is an urgent need for having this documentation available on demand, i.e., on-line.

In Sec. II of this report, some of the problems encountered with present procedures for documenting programs are discussed, and an attempt is made to categorize the different types of documentation according to the detail of the information. Section III describes CTSS and Sec. IV describes the INFO system, a model for an on-line documentation system, indicating the objectives of the system and directions for its use. Section V discusses some of the considerations used in implementing the information system. Section VI describes the COMIT programming language and how some of its features are used in the INFO system. Section VII suggests some further modification to this on-line storage and retrieval system, and Sec. VIII summarizes the research work.

II. PROGRAM DOCUMENTATION

The documenting of computer programs has been a problem since the development of the programmed computer (Ref. 2). Whenever a computer program is written or a programming system developed, it must be described by a set of documents which satisfy the needs or curiosity of the various people who desire to use or modify the program or system. The type of documentation desired varies, depending on one's interest. To the very uninformed, the title or name of the program may be sufficient. On the other hand, to a person who wants to make a change in a program, a listing of the program in the original programming language is required. Thus, depending on one's purpose, various forms of documentation are needed.

Even though a computer is a finite state machine with a finite memory and has limited computation power (assuming a limited running time), a vast number of computation algorithms can be written by specifying a sequence of computer instructions. Many of these sequences, in the form of subroutines, can be included in different computation algorithms. Thus, once a

programmer solves one problem, the techniques used (the routines used in a program) can often be useful in solving another problem.

In order for a program written by one programmer to be useful to another programmer, it must be documented in such a way that the second programmer can use the program without first analyzing it. This means that the documentation should clearly state the calling sequence of the routine, giving the format of input parameters, the process the routine performs on the parameters, the format of the resultant output parameters, and the various exits from the routine. Programmers should be encouraged to document their work and should be permitted to devote the necessary time and effort required to provide meaningful documentation. Unfortunately, the attempt to document a set of programs is rarely successful; consequently, it is often easier to rewrite a program than it is to analyze one already written to determine how to use it.

Computer programmers, like most scientists, are exceedingly demanding in their quest for written information, but are reluctant to provide written information about their own work. This is not to say that programmers are selfish and want to keep their work to themselves, but rather to point out the difficulty in satisfying the demand for documentation. Good written documentation is provided by programmers who have a public spirited attitude and pride in their work. Once the documentation of a program is provided, it is up to the interested person to read and study the documentation.

Knowing how to do something requires more than just documentation. Many problems, such as lack of knowledge, are blamed on poor documentation. No amount of documentation can replace the thinking process and the effort required to learn about the work of others. The problem of keeping aware of current information is not an easy one. Nevertheless, the advent of larger computers and more sophisticated programming systems makes it highly desirable to improve communications among programmers in order to minimize redundant effort.

Once the desired documentation of a program is provided, preferably by the original author of the program, the documentation must be distributed or made available to those who want to be informed about the program. Thus, the problem of documenting computer programs may be separated into two parts: (1) How to prepare the documentation, and (2) how to make the documentation available to the right people at the right time. First, we shall consider how to prepare the documentation of a program, then we shall discuss ways of disseminating program information.

In discussing how to prepare the documentation of a program, we must indicate the type of description needed and how this type is to be prepared. Since different people have different requirements, a program must be described in many ways. Various readers require different kinds of documentation to varying amounts of detail. The complete documentation of a program might be provided by the following ten document items tailored to the needs of different people.

(1) Program Name: – This alone might be sufficient to indicate that a program is or is not of interest.

(2) Classification: – If a short title or name is not sufficient to indicate the general class of problem with which a program is concerned, then a set of related titles, key words, or descriptors would be useful.

(3) Sentence Description of When Used: – A few sentences indicating the context in which a program might be used would supplement the information supplied by the classification descriptors.

(4) Paragraph Description of What Is Performed: – A paragraph describing what a program does should give all the information a user would need to determine whether or not he wished to use the program. It is here that the purpose of a program is indicated.

(5) Program Usage: – A description of how a program is used might include a calling sequence if it is a subprogram, parameters if it is a command program, or operating instructions if it is a main system. If the program itself is a versatile system, this might indicate some other extensive manual as a reference.

(6) Program Information: – This information might include such items as:

(a) The author of a program

(b) A mailing list of people interested in the program

(c) Amount of storage required

(d) System symbols or entries to the program

(e) Other subprograms which the given program uses

(f) Data common to other programs

(g) Speed of execution

(h) Programming language in which it is written

(i) Precision in which numerical data are calculated

(j) Type of program, i.e., pure procedure, generalized subroutine, recursive subroutine, etc.

The following types of documentation could be helpful in determining the algorithms used by a program:

(7) Page Description of Process: – In this item of documentation, the process performed by the program is described. Details are included which are not essential to its use but which would concern those making modifications to the program.

(8) Flow Chart of Procedure: – Flow charts in the form of block diagrams describe the procedures carried out by a program. The level of detail shown by a flow chart varies to some degree, but it is here that the implementation of a process might be first determined.

(9) Description of Flow-Charted Process: – This includes the description of algorithms used and how they are implemented. If a program is written in a way that makes use of particular computer characteristics, e.g., the bit format of a computer word, this fact must be described in detail. This type of documentation is most detailed and includes all phases of a program's coding. It is quite often eliminated, since it is here that a great amount of time and effort is required for documentation.

(10) Program Listing: – The most detailed description is the program listing, written in a commonly understood programming language. Listings are easy to produce but substantial effort is required to determine from this type of documentation what a program does and how it is to be used. It is true that some programming languages are more easily understood than others (e.g., MAD rather than FAP), but nevertheless, the reader is presented with a myriad of details so that the overall operations of the program are buried within the details.

Note that in items (3), (4), and (7) the length of the documentation (i.e., sentence, paragraph, or page) is used as an indication of the amount of detail presented. Program Information, item (6), lumps together all miscellaneous items which might be pertinent to some programs, but not to others. If these ten items of documentation were provided for all programs, a certain amount of standardization would be achieved and the needed information might be available. Even assuming that programs were documented as set forth, the problem of making the documentation available at the time it is needed still exists.

Let us consider the problem of how to prepare this documentation. Only the author of a program is really qualified to describe what the program does, but quite often he is more interested in writing or thinking about new programs than in spending his time documenting what he has already done. Also, a good programmer is not necessarily a good manual writer. If the documentation of a program is to be easily understood by a user, then the documentation should conform to certain minimum standards of format and types of items that must be included in the documentation. Thus, there arises a need for a system to help authors document their programs. (A system for this purpose has been developed for documenting the programs associated with CTSS and will be described in Sec. IV).

In addition, there is the problem of how to make the documentation available to the right people at the right time. There are two types of problems with which a user is concerned: The first is determining in which program or programs he is interested, and the second is obtaining information about a specific program.

Consider the problem of determining the program or programs of interest. If the programs in the system were classified by using category descriptors organized into a hierarchical tree-like structure, a user could examine categories of programs until he found one that included the program or programs in which he is interested. For example, a user might request the sub-descriptor categories under the descriptor "COMMAND" and obtain a list of descriptors used to classify commands. The user may then list the names of the programs under a particular category, e.g., under the descriptor "EDITING". Although it is not necessary that the set of descriptors form a hierarchy, the useful concept of sub and super categories might be helpful in isolating a particular program. The problem of category retrieval is based upon a useful assignment of descriptors or key words to a program and the lumping together of descriptors into more general classifications. No suggestions are being offered on how to generate a set of useful descriptors or how to incorporate changes in the structural organization of descriptor categories.

Another type of retrieval problem is centered around miscellaneous program information (item (6) of the previous list). A user of a system might want to know which subprograms are used, i.e., called by a given program, or how much space is required by a program and all those programs which it calls. This information would be useful when incremental changes to a program result in small, but persistent, increases in the program size. Consequently, the program may take up more space than is available. Conversely, a user might want to know which programs call a given program or a particular entry to a given program. This information is useful when a change is contemplated in the given program and one is interested in the consequences of such a change. Other types of retrieval request which a user might like to ask would involve finding the programs written by a particular author or written since a given date. When a user wants information about a particular program, he may want all the available documentation, or he may want only one item of information, e.g., its usage. This need is most common to on-line users of a computation facility who forget the calling sequence of a program and would like to find this information as soon as possible and with the least amount of effort.

Thus, we have indicated the types of documentation that are needed and the types of retrieval requests that a user might want to make. It should be pointed out that the problem of retrieving documentation on computer programs is similar to the library problem of general information retrieval, although it is quantitatively different. The library problem is caused by the enormous proliferation of published material and the difficulty of relating one article or book to another.

In solving the library problem, the information scientist, using techniques of analysis, attempts to relate or categorize items for retrieval on the basis of specific criteria.

On the other hand, the problem of documenting computer programs is concerned with specifying information in such a way that it can be communicated from one person to another. The information scientist is free to organize and specify the information as he deems best, so that it can be understood by someone unfamiliar with the information. It can be said that to some degree the library problem is one of analysis, whereas the documenting of computer programs is one of synthesis. The central problems are that the library information must be analyzed to determine related items and the computer information, once synthesized, must be distributed to the right people. Both attempt to provide all pertinent and useful information without creating a flood of written material.

The Compatible Time-Sharing System provides a means of attacking the retrieval problem through interaction with an on-line user and, in addition, provides a means of uncovering the particular needs of the on-line programmer. The problem of program documentation is part of the general information retrieval problem and must contend with the problems associated with storing a large body of information and searching for items which satisfy a particular need. By restricting the information base to that of the computer programs associated with CTSS, the problem becomes quantitatively more manageable.

III. THE COMPATIBLE TIME-SHARING SYSTEM

A. Computation Facility

The Compatible Time-Sharing System (CTSS) is a programming system for a configuration of computer hardware centered around the 7094 computer. The hardware consists of a 7094 computer with two 32k word core memories; tape, drum and disk storage; and a 7750 communications computer which handles input/output messages to and from remote consoles. A programmer at a console communicates with the system through a set of commands. These commands allow the user-programmer to write and run programs using standard procedures with the added feature of having on-line interaction with his program. Through the use of the command language, a user can perform many tasks which make the computation facility more accessible.

In addition to the command programs, the time-sharing supervisor contains programs associated with the auxiliary storage devices and the system as a whole. These routines perform for the user of the system the necessary code conversions, buffering, and accounting which the hardware devices require. These tasks, performed by the time-sharing supervisor, are called upon by special entries to the supervisor. The set of supervisor entries are, in essence, a set of routines which the programmer can make use of in his programs. The user-programmer can also draw upon a standard library of subprograms for direct inclusion in his computation algorithms. These library subprograms may in turn call upon the supervisor routines to handle input/ouput functions.

Besides the system commands, supervisor entries, and library subprograms, a user has access to a set of programs which are stored in a public file. Thus, from a users point of view, the four types of programs associated with CTSS are: (a) System Commands, (b) Supervisor Entries, (c) Library Subprograms, and (d) Public Programs.

B. Documentation of CTSS

The documentation which is presently available to describe the programs associated with CTSS consists of (Ref. 3):

(1) The Programmers Guide
(2) Time-Sharing System Notes
(3) Computation Center and Project MAC Memos
(4) CTSS Bulletins.

Unfortunately, the Programmers Guide contains false and out-of-date information, and consequently it is not adequate documentation of CTSS. The Bulletins, Memos, and Notes are of some help but, since they are not indexed and are written at different levels of sophistication and detail, something more is needed. Some Bulletins indicate that a new command or system feature is available, but do not indicate how they are to be used. Other Bulletins indicate changes or modifications to programs that were previously described but do not indicate the documentation that has become superseded or outdated. The major criticism with the present documentation is that many aspects of the system are not described at all.

On-line use of a computer is more versatile than the conventional use of a computer through batch processing. With this on-line ability come many new ways of employing the computation facility. As new features are developed, information on these new features must be communicated to the users of the on-line facility. Thus, the problems of documentation associated with an on-line facility are greatly increased and a new approach to this problem is required.

Adequate documentation should include a description of programs written by the users as well as by the system's programming staff. Many users have either written programs which are of general use or have developed techniques of on-line programming (such as useful RUNCOM chains) which should be made known to all users. The following paragraphs from a report on Project MAC and its users (Ref. 4) emphasize the need for More Adequate Communications (MAC) between on-line users.

> "In a broader sense, documentation refers not only to a description of CTSS operations, but to communication among users about mutually useful programs. Such communication is virtually non-existent at MAC, except by word of mouth among members of related subgroups. The result is an enormous duplication of effort. For example, an appreciable number of users have independently written programs which produce "typewriter graphs" – curves printed by appropriate spacing of teletype/typewriter characters. As another example, various users have written mathematical service programs for various standard computations. These duplications are especially striking because one of the chief reasons for preferring a large time-shared computer to an ensemble of smaller machines is that it permits the users to enjoy the fruits of one another's labor. This potential advantage is not being realized at present.
>
> "Examples of inadequate documentation can easily be multiplied. One mathematician who tried to use the system eventually gave up because he was such a poor typist that he had to enter almost every line of program repeatedly. He said he would not consider using CTSS again until there was an editing routine that permitted changes of single characters within a line. At the very time of his complaint, Samuel's editing routine, which incorporates this feature, was available and being widely used. Since then it has become even more readily available as a system command, but even now this user would have no way of finding this out, if he were to make another attempt."

In fairness to the documentation of CTSS, one should note that for a system which has been designed and implemented in a relatively short time, CTSS is rather well documented. In fact,

numerous memos have been written which indicate the philosophy of time-sharing and describe the design of the supervisor system (see Ref. 5). Since CTSS is used by many people who are eager to have new features implemented, it is understandable that the system's programming staff has spent their efforts improving the system rather than completing the documentation. Nevertheless, the system would be improved if only the documentation were better. Another point of importance is that CTSS has been designed as an experimental system, and thus does not have the requirements for complete documentation, although as a research project it should.

Now assume that a new reference manual was prepared which contained the documentation of the system as it stands today. If this new manual were well written, well indexed, and contained the documentation of all present programs, other problems in documentation would still exist.

First of all, this new manual would have to be distributed to all users, novices as well as experienced users of CTSS. The novice would find the amount of information overwhelming. He would read the manual to learn enough information so that he could begin to use the system. As the novice spends more and more time at his on-line console, he tends not to go back to the reference manual unless he has a specific problem. Eventually, after much frustration, he becomes like the experienced user of today, who has read or heard of many features of the system but is inclined to use only the techniques that he has used many times before. One often avoids using a particular program because he never learned how, or forgot how to use it, or doesn't have the documentation at hand. Hence, the conclusion to be drawn is that it is important to have the right documentation available to the on-line user.

A second problem which influences the documentation is that programs are modified or improved in such a way that the original description becomes out of date. This requires that memos be re-circulated to notify users of the changes made. Since not everyone will want the same amount of detail, a single memo must contain different levels of description, or many memos must be prepared for distribution to either programmers, supervisors, operators, or administrative personnel.

Present procedures for distributing memos require that a programmer either request documentation on a particular program, or that he be included on a distribution list for a particular category of documentation. These procedures fail in the following ways:

(1) The right people do not always get the documentation of interest. Either they do not know that the documentation exists in order to request it, or they are not on the right distribution list.

(2) The user does not keep an up-to-date index of the documents that he receives and hence does not know what is available to him.

(3) The user cannot find the documents that were distributed and hence must request additional copies. This creates more problems, since additional copies may not be available, and consequently more may have to be duplicated, forcing the user to wait before he can get the desired information.

(4) Programs are changed or modified and new or amended documents are not written and distributed.

(5) Even if amended documents were written, it would be difficult to know who must be informed of the changes. That is, the distribution lists must be kept up to date.

(6) There are inherent delays in providing the documentation of a program, or of a modification of a program, which would prevent a user from obtaining the latest information available (e.g., delays which occur during preparation of a memo or during its distribution).

A third problem in keeping up-to-date documentation arises when on-line consoles are physically removed from the computation facility. For example, the users at Lincoln Laboratory have little contact with the administrative staff and have no contact with other users. Any system of documentation from a computation center to remote users is bound to be somewhat unsatisfactory and at best is bound to impose delays caused by external delivery services.

The advancement of the art of time sharing makes it feasible for many programmers to use a computer from an on-line console. It also becomes reasonable to have many on-line users who are physically dispersed and whose only communication with the computer is through the on-line consoles. The problems of documentation then become a major and not a minor problem. Good written documentation is then more important, since there is no person with whom to consult. Unless the classical system of documentation is perfect, a near impossibility, another method of obtaining needed information is required. Also, the time that the documentation of a program is needed is when a programmer wants to use a program, that is, while he is sitting at his console. Unless his console is in his office, he will find that he does not have easy access to the needed documentation. The problem of remoteness from the computation facility thus increases the need for good documentation.

In order to meet the requirements of good program documentation, a system must be developed to provide, in a better way, the right documentation when it is needed. Hence, a standardized and more accessible documentation procedure for CTSS is needed. An on-line documentation system offers the best means of providing information to CTSS users.

The on-line documentation must be integrated with the other forms of documentation and, in particular, the comprehensive Programmer's Guide which describes all facets of CTSS. Since the original Programmer's Guide (Ref. 1) was published, the facilities of CTSS have grown tremendously, as one could expect. The Programmer's Guide, now in the process of being re-written, is intended to serve as an introduction to all phases of on-line computation with CTSS and as a reference manual for those who are already experienced users. The on-line documentation system is intended to supplement the more comprehensive reference manual, providing information on newly written programs as well as information on programs which have long been in use. The on-line system also enables searching the whole body of available information. The system may be useful in determining the nature of new programs just added to CTSS as well as to remind a user of the usage or calling sequence of programs which are less widely employed.

IV. THE ON-LINE DOCUMENTATION SYSTEM

A. Objectives

In order to better provide information about the programs associated with the Compatible Time-Sharing System, an on-line system for documenting computer programs has been developed. The design of this on-line system attempts to satisfy the following objectives:

(1) Have up-to-date information available to the user on request, thus eliminating the delays which occur in any memo distribution system.

(2) Have the ability to obtain specific information on request, e.g., the author of a routine, as well as the complete documentation of a routine.

(3) Give textual output in steps, i.e., printed according to information item types (1) through (6) as described in Sec. II, indicating the amount of printing that will result.

(4) Provide the facility to search through the library of programs to determine the ones which satisfy particular conditions.

(5) Standardize the format of the program description by requiring that when a new program is added to the system all information of interest is provided.

(6) Permit editorial control of the program documentation that is to be included in the on-line system.

The on-line documentation system has been implemented as a CTSS command with command name INFO. Upon execution of this command, requests can be made to obtain answers to the following types of questions:

(1) What does the command STRACE do, i.e., when could it be used?

(2) How is the supervisor entry .FILDR used, i.e., what is its calling sequence?

(3) What new programs have been added to the TSS library since September 1st?

(4) Who is responsible for the command GPSS?

(5) What are the names of the command programs written by the STAFF since August 1st?

The system has been designed as a general-purpose means of storing and retrieving textual information about computer programs. The immediate objective is to provide documentation of system commands, supervisor entries, library subprograms, and public programs. These types of programs have been chosen because there is an urgent need for having this documentation available on demand, i.e., on-line.

The information describing a program is divided into information items. Each item of information is associated with an item name and is referred to as the item value of the associated item name. For example, the item value "WINETT" is associated with the item name "AUTHOR." The following items of information indicate what is required as documentation of a computer program:

(1) Program NAME (N) – A single word.

(2) Program TYPE (T) – One of the following: COMMAND, ENTRY, LIBRARY, or PUBLIC.

(3) DESCRIPTORS (D) – Key words used to classify the programs in the information files.

(4) PURPOSE (P) – A short abstract or sentence description indicating the context in which a program might be used.

(5) USAGE (U) – The instructions of how to use the program, e.g., the calling sequence.

(6) Programming LANGUAGE (L) – The language in which the program is written.

(7) REFERENCE (R) – A bibliography of where more information about the program may be obtained.

(8) AUTHOR (A) – The name of the person who is responsible for the program.

(9) DATA (DA) – The date the information was last entered or modified.

Additional information items may also be defined, e.g., program size, transfer vector, etc., but the above items are considered required to document any program.

B. System Usage

A model of this information system has been implemented as a CTSS command program with command name INFO. The system may be initiated as a console command or may be "chained to" from another program. If, when the INFO system is called, the NAME of a program is given as a command parameter, the documentation on that program will be printed, after which the system will call CHNCOM. This procedure allows other command programs to have access to their documentation. For example, when no parameters are supplied with a command which requires at least one parameter, the command should chain to the INFO command with the command name as a parameter. This technique would provide a means of tying the documentation of a command program to the command itself.

If only the command name INFO is typed, the system will respond

TYPE REQUEST, OR C.R. FOR INSTRUCTIONS..

whereupon a carriage return will initiate the request to describe the INFO command.

Alternatively, requests can be typed to the INFO system. There are three classes of requests: (a) Retrieval requests to obtain information from the system, (b) Storage requests for adding, changing, or deleting information from the system, and (c) System requests which affect the operation of the system. The Retrieval requests – DESCRIBE (D), LIST (L), and FIND (F) – are to be used by all CTSS users. The Storage requests – STORE (S), EDIT (E), ALTER (A), and REMOVE (R) – are to be used by the people responsible for the information stored within the system. This responsibility may be shared with special users as will be discussed in Sec. VI. The System request – QUIT – is used to terminate communications with the system, and the requests – END, TSSFIL, and USRFIL – are used for changing the operation of the system.

Whenever the INFO system prints a comment followed by two periods, it is the user's turn to type. After processing each request, the system types

OK..

To obtain a description of a Storage or Retrieval request, the user types the request name only. A request to the INFO system indicates three types of semantic information: (1) an imperative request to the system, (2) a list of single information words, or (3) information items specified by item names together with the item values associated with the item names. A request to the system is assumed to be indicated by one of the first few words typed. Other words following the request name may be item names which are added to a list of "information words" or may specify the values of information items which are added to a list of pairs consisting of an item name and its value. When either the word "IS" or "ARE" is encountered in a request, it is assumed that the previous word is an item name and that the following words form the item value. The input specifying the item value must be terminated by a comma (or the carriage return at the end of the request) and may be followed by other item names and their values or by item names alone. If the word "THEN" appears as an information word, the input scanned so far is assumed to constitute a request. After the request is processed, the input following the word "THEN" is scanned for the specification of another request. Thus the word "THEN" indicates the termination of a request and allows multiple requests to be typed. Words other than item names or item values or the word "THEN" may be typed but are ignored by the system.

Requests and item names may be abbreviated by their first letter (except the item name DATE which is abbreviated DA). If an item value is specified more than once in an input request,

the value last specified takes precedence. Thus, the on-line user may change or correct the specification of an item value by retyping the item name together with the item value in the same input request.

To continue the typing of a request on another line, precede the carriage return (C. R.) by a dash (–). When in doubt of what to do, type a carriage return.

RETRIEVAL REQUESTS

1. The DESCRIBE (D) request:

DESCRIBE NAME IS name, i(1) ... i(n)

This request is used to obtain the documentation of a program whose name is known. The input with this request gives the program name and the names of the desired items of information. If no item names are specified, the information on all items will be printed. For example,

DESCRIBE THE COMMAND WHOSE NAME IS INFO

produces all the documentation associated with the INFO command, and

D N IS INFO, USAGE

prints just the item USAGE for the INFO program.

When more than five lines of text are to be printed, the INFO system informs the user of the number of lines which follow. After realizing how much information will be printed, the on-line user may terminate the request by pressing the CTSS interrupt or quit button.

If the interrupt button is pressed the user may type "CONTINUE (C)" to resume printing or "RESTART (R)" to type another request. Printing will be resumed approximately ten lines after the line at which printing was interrupted. (This is due to the fact that the CTSS output buffers are cleared on interrupt.) Since a number of lines are lost on interrupt, the process of interrupting and continuing provides a means of skipping lines of documentation. Unfortunately, this procedure gets very poor response from CTSS.

If the quit button is pressed, the on-line user may type another command or type the CTSS command "START" to continue as described above. This procedure gets very much better response from CTSS.

2. The LIST (L) request:

LIST TYPE IS type, i(1) ... i(n)

This request is used to obtain a list of the names of all information items, a list of the values of certain information items, or to list the names of all CTSS programs of a particular type. The request may ask for the values of one or more of the following items to be listed:

ITEMS, AUTHORS, DESCRIPTORS,
LANGUAGES, TYPES, or NAMES

or may also request a list of all CTSS programs of a particular type by typing one or more of the types

COMMAND ENTRY LIBRARY or PUBLIC

after the words: TYPE IS. The list of programs of a particular type are obtained directly from CTSS and thus automatically provide the most relevant list of programs available.

A request to

LIST NAMES

causes a list of the programs of all types to be printed. A list of descriptors may be obtained by typing

LIST THE DESCRIPTORS

or just L D

3. The FIND (F) request:

FIND i(1) IS v(1), . . . , i(n) IS v(n)

This request is used to perform a search for the program or programs which have particular information item values. The items to be matched are given by typing the item names together with their item values. Acceptable items for searching are:

TYPE, DESCRIPTORS, AUTHOR, DATA, and LANGUAGE.

A date value must be given in the form – DATE IS mm/dd/yy where mm is a numerical month, dd is a numerical day, and yy is a numerical year. All programs whose date is greater than that given will be printed, i.e., the most recently documented programs. Descriptors are single words typed in any order and separated by spaces or the word AND.

For example, to find the commands which were documented since September 1, 1964 and have at least the descriptors UTILITY and EDITING type –

FIND TYPE IS COMMAND, DATE IS 9/01/64, DESCRIPTORS –
ARE UTILITY AND EDITING

or F T IS C, DA IS 9/01/64, D ARE UTILITY EDITING .

(Note the use of the dash to continue the input request on the next line.)

When a search results in more than twenty matching items, the system asks whether the user wants to continue the search. The user may then type YES or NO. For each twenty more matching items, the user is given the option of continuing.

STORAGE REQUESTS

4. The STORE (S) request:

STORE NAME IS name, FILE IS file, i(1) IS v(1), –
. . . , i(n) IS v(n)

This request enables one to enter information about a new program into an information file. This request requires that information values be provided for each required item in the form:

item name IS/ARE item value .

The NAME of new information items may be defined by typing the new item name and its value. When the INFO system prints an item name followed by two periods, the user is to type the value of that item. Item names and item values of other items may be supplied following the item value which was requested by typing a comma after each item value and thus anticipating the required input and reducing on-line interaction.

If the word FILE is specified in the input specification, a file with primary name the same as the program name (if specified) and secondary name INFO is read. This file is assumed to contain item values for this program where each item value is preceded by a line giving the item name prefixed by a period and beginning in column one. If the primary name of this input file is not the same as the program name, the file name may be specified by typing the item

FILE IS file name.

If a file name is specified and a program NAME is not specified, the NAME of the program may be read from the input file. A program NAME is indicated in an input file by the presence of two periods before the program NAME. An input file may specify the documentation of many programs by preceding the documentation of each program with a line giving the program NAME prefixed by the two periods (e.g., . . INFO). The priming of the command documentation was done from an input file (with name COMAND INFO) of this type by typing

STORE FILE IS COMMAND .

5. The EDIT (E) request:

EDIT NAME IS name .

This request re-creates a BCD file (as a line marked file) from the information in the system for use in making changes to information items using some CTSS editing procedure. The EDIT request requires that the program NAME be specified. Each information item is preceded by a line giving the item name prefixed by a period (e.g., . PURPOSE), and consequently no line of an item value should begin with a period. The primary name of the file created is the same as the program name and the secondary name is INFO.

6. The ALTER (A) request:

ALTER NAME IS name, i(1) IS v(1), . . . , i(n) IS v(n)

This request allows one to change item values in the information documenting a program or to store additional information items. The ALTER request requires that the program name be specified and is used like the STORE request. The ALTER request is different from the STORE request in that it does not require that values for all information items be specified. That is, the user-system interaction is different.

7. The REMOVE (R) request:

REMOVE NAME IS name, D IS d, A IS a, I IS i

This request is used to delete an AUTHOR, DESCRIPTOR, or ITEM name from the appropriate list, or to delete the documentation of a program from an information file when a program is deleted from CTSS. To REMOVE the documentation of a program, give the program NAME. To REMOVE an AUTHOR from the list of AUTHORs or a DESCRIPTOR from the list of DESCRIPTORs, specify the item value to be removed. To REMOVE an ITEM name from the list of ITEMs, specify the ITEM name. Verification of each request to remove the documentation of a program is required.

SYSTEM REQUESTS

8. The QUIT (QU) request:

This request causes the INFO system to call CHNCOM, and it may be used to terminate the

INFO command or to chain to other commands.

9. The END request:

This request causes the INFO system to terminate through the standard COMIT termination sequence. (The INFO command has been written in the COMIT language.) The amount of unused free storage, i.e., the number of WORKSPACE registers, is printed. This request may not be abbreviated.

10. The TSSFIL request:

This request causes the INFO files to be obtained from one of the CTSS system file directories and is issued before thc INFO system is included as a CTSS command. This request may not be abbreviated.

11. The USRFIL request:

This request causes the INFO files to be obtained from the user's file directory rather than the system file directory. This request may be employed by a user to indicate that the documentation files are to be obtained from the user's file directory. In this way, a user may keep documentation of his private programs. This request may not be abbreviated.

C. User-System Interaction

The INFO command responds to requests typed by the user by either performing the desired request, printing a comment, or asking a question. When only the command name INFO is typed by the user the system responds with

TYPE REQUEST, OR C.R. FOR INSTRUCTIONS..

whereupon the user may type a request or a carriage return for instructions of how to use the INFO command. After completion of a request, the system responds with

OK..

and the user may typc another request. Whenever the system terminates a comment with two periods, "..", it is the users turn to type next. The two periods can be interpreted either as a final period or as a question mark.

The response from the INFO system may occur when it is interpreting a request or when it is processing a particular request. The following responscs from the INFO system may occur:

1. On Input –

   (a) ___ IS A NEW INFORMATION ITEM, CORRECT IT OR TYPE OK OR IGNORE..

   (b) ___ IS A NEW DESCRIPTOR, CORRECT IT OR TYPE OK OR IGNORE..

   (c) ___ IS A NEW AUTHOR, CORRECT IT OR TYPE OK OR IGNORE..

The user may correct a misspelled word, type OK to indicate that the word should be accepted, or type IGNORE to continue processing the request. In this way, the user is notified when he is

adding to the information for which a search can be made.

(d) NO ROOM FOR NEW ITEM _____, PLEASE NOTIFY STAFF..

This comment is printed when too many new information items have been defined. The system presently permits thirty new information items.

(e) ___ IS NOT A TYPE, CORRECT IT OR TYPE IGNORE..

(f) ___ IS NOT A LANGUAGE VALUE, CORRECT IT OR TYPE IGNORE..

Only one of the set of pre-specified values for the information items TYPE and LANGUAGE is permitted, in order to simplify the storage of this information and to facilitate searching. Note that these information items can take on only a given set of values; whereas, new values for other items may be defined by the on-line user. The set of pre-specified values may be changed or enlarged by a trivial change to the INFO command program.

(g) ___ IS NOT IN THE FORMAT FOR DATE, FORMAT MUST BE MM/DD/YY..

A date must be given as MM/DD/YY where MM is a numerical month, DD is a numerical day, and YY is a numerical year.

(h) REQUEST NAME MISSING. REQUESTS ARE – DESCRIBE, LIST, FIND, STORE, EDIT, ALTER, and REMOVE..

If a request name is not found among the information words and at least one item name together with an item value is specified, it is assumed that the request name was misspelled or not typed, and the on-line user is requested to specify a request name. Any information word previously typed will have been ignored and must be retyped, but information values do not have to be retyped.

(i) FILE_____BEING READ.

(j) FILE_____BEING READ TO OBTAIN ITEMS FOR_____ PROGRAM.

These comments are printed when an input file is read to obtain the values of information items.

(k) FILE NAME NOT GIVEN, FILE IGNORED.

2. On DESCRIBE –

(a) PROGRAM NAME IS..

The DESCRIBE request requires that the name of the program to be described be specified. The user should type the program name and, if desired, the information items to be printed.

(b) ___ NOT DOCUMENTED.

This comment is printed when the documentation of the program requested has not been stored.

(c) (_____LINES FOLLOW)

When more than five lines of text are to be printed, the INFO system informs the user of the number of lines which follow.

(d) PROGRAM_____NOT IN INFO FILE _____, PLEASE NOTIFY STAFF.

This comment indicates an error which might have been caused by a CTSS system failure.

3. On LIST –

(a) ___ IS NOT AN ACCEPTABLE ITEM FOR LIST.

Only the following items may be listed: ITEMS, DESCRIPTORS, AUTHORS, LANGUAGES, TYPES, and NAMES. If one requests that NAMES be LISTed, the system will obtain from the CTSS supervisor the list of programs (commands and subprograms) for each TYPE. This list will indicate the most recent status of the CTSS programs, since the list is obtained from the supervisor itself.

(b) NO ___ 'S TO LIST.

This comment is printed if no DESCRIPTORS or AUTHORS have been defined.

(c) ___ IS NOT A DESCRIPTOR.
(d) ___ IS NOT AN AUTHOR.

In preference to listing all AUTHORS or all DESCRIPTORS, a user may request to list the name of a particular AUTHOR or DESCRIPTOR to check whether it has been defined. One of the above comments is printed if the particular value requested has not been defined.

(e) ___ IS NOT A COMMAND PROGRAM.
(f) ___ IS NOT AN ENTRY PROGRAM.
(g) ___ IS NOT A LIBRARY PROGRAM.
(h) ___ IS NOT A PUBLIC PROGRAM.

One of the above comments may be printed if a particular program does not exist as one of the programs associated with CTSS. This information is obtained from the CTSS supervisor and has no relation to whether it has been documented or not.

4. On FIND –

(a) NO MATCHING ITEMS FOUND.

This comment is printed as the result of a search for items with specified item values.

(b) ___ ITEMS FOUND.
(c) ___ ITEMS FOUND SO FAR, DO YOU WANT TO CONTINUE. .

When a search results in more than twenty matching items, this comment is printed before the twenty items found are listed. The user must type YES or NO.

5. On STORE –

(a) ___ IS. .

Certain information items are required for the documentation of a program to be stored in the INFO system. For each one of these, a comment of the above form will be printed whereupon the user is expected to provide the requested item value. If the user terminates the value with a comma, he may continue to specify other item values for this STORE request. For example, after the system types –

TYPE IS . .

the user may type

COMMAND, DESCRIPTOR IS UTILITY, AUTHOR IS STAFF

to continue specifying the item values for the store request.

(b) ___ IS ALREADY STORED, DO YOU WANT TO ALTER. .

If an attempt is made to store the documentation of a program that has already been stored, the on-line user has the option of ALTERing the documentation for that program with new information.

6. On EDIT –

(a) PROGRAM NAME IS . .

A program name is required with the EDIT request.

(b) ___ NOT FOUND.

This is a possible response from the EDIT request.

7. On ALTER –

(a) PROGRAM NAME IS . .

A program name is required with the ALTER request.

(b) ___ NOT DOCUMENTED, DO YOU WANT TO STORE . .

If one attempts to modify the documentation of a program that has not been documented, he has the option of storing the complete documentation for that program.

8. On REMOVE –

(a) ___ NOT FOUND.

The response if a program is not documented, and hence cannot be deleted.

(b) ___ IS NOT AN AUTHOR.

(c) ___ IS NOT A DESCRIPTOR.

(d) ___ IS NOT AN OPTIONAL ITEM NAME.

One cannot type an AUTHOR or DESCRIPTOR in a request unless it is defined in the appropriate list.

(e) ___ IS BEING REMOVED FROM THE FILE, OK . .

Verification is required before the documentation of a program can be removed from the information file.

D. The Data Base for the INFO System

In order for this INFO system to satisfy the objectives of providing on-line documentation of the programs associated with CTSS, the system must be primed with meaningful information. This is no simple task. The files containing the documentation information must be made available to the INFO system. Since the documentation consists of textual information, it must be prepared for storing in the system either by the authors of the programs or by some other knowledgeable person. In addition, once the documentation has been brought up to date, i.e.,

information stored on the present set of programs, it must be kept up to date.

Consider first the problem of priming the system with information on programs already available for use with CTSS. These programs eonsist of system eommands, supervisor entries, library subprograms, and public programs. The system has already heen primed with documentation of the 82 system commands in the concise format desirable for the on-line retrieval system. The ED command, which permits input and context editing of a BCD file, was used to produee a file (with name COMAND INFO) containing the documentation of the commands. This file was then used in a STORE request to store the doeumentation on COMMANDS. Following similar procedures, on-line documentation for the other types of programs must he provided. It is suggested that this be done as the new reference manual is being prepared. This task might be assigned to a system's librarian or to one of the staff eonsultants.

Now consider the problem of keeping the doeumentation information up to date. This entails providing additional documentation when new programs are added to the system and providing revisions to the documentation already stored when existing programs are modified. In an ideal situation, documentation would be automatically obtained and no human supervision would be required. Since the doeumentation of programs consists mostly of textual material, it must be written by a knowledgeable person. On the other hand, an up-to-date list of the different types of available programs ean be automatically obtained by using the LIST request.

One of the objectives of the on-line system is to aid in providing the desired doeumentation. The STORE request requires that certain information items be provided, and in this way an attempt is made to standardize the documentation. Each time the documentation of a program is stored or altered, the INFO system automatically supplies the information value to the information item DATE. That is, the date the doeumentation is stored or altered is automatically stored and the on-line user does not have to input its value. If each author of a program were to store the documentation of his own programs, the INFO system could also provide the value to the information item AUTHOR by obtaining this information from the CTSS supervisor (the supervisor knows the name of a user along with his problem and programmer number).

This brings up the question of responsihility, i.e., who should he responsible for keeping the INFO system up to date and accurate. The information files associated with the INFO eommand are to he stored with the system files (i.e., in the directory of M1416, common file 2) but are now temporarily stored with the public files (COMFIL P is in the directory of M1416, common file 4). This restricts the numher of people who modify the information to those who are assigned an M1416 problem numher. Thus, the machinery which is presently built into the CTSS supervisor is used to control who is permitted to alter the information files. This does not require that a staff member write the documentation of all programs. Any user who writes a system program (COMMAND, ENTRY, LIBRARY, or PUBLIC program) may be asked to provide an INFO file in the form appropriate for the STORE or ALTER request, and this file can then he used by a privileged user (problem number of M1416) as input to the INFO system.

The question of who has editorial control of the documentation, i.e., who has responsibility for the INFO system, is still not answered. No simple answer is apparent. All that can be said is that a system lihrarian who is responsible for all forms of documentation must be given the responsibility of monitoring the on-line system.

The following procedure, which appears to be feasible, would help the system's librarian keep up-to-date documentation on programs which are developed and eontinually modified by a

special user group. If INFO files are prepared by the special user groups for the programs for which they are responsible, these INFO files could be obtained from the special user group's file directory and used as input to the INFO system. This procedure could be performed by a special-purpose system program. This system program to update the documentation could be run automatically at specific times during operations of CTSS. In this way, the user group which develops a program is also given the responsibility of updating the documentation of the program. All that is required is that an INFO file be created and included in the user groups file directory. This INFO file would be processed in a similar manner as REQUEST FILE's are now processed. The special system INFO updating program would have to know, for each program, which user was permitted to update its associated documentation.

The problem of keeping the on-line system up to date with information on new programs will always be with us, as will all problems concerning documentation. It is hoped that the existence of an on-line system will tend to centralize the effort. Even without altering the problem of preparing documentation, an on-line system will help to make the documentation more readily available to the on-line users.

V. DESIGN CONSIDERATIONS

A. General Approach

To insure that an on-line documentation system continues to be useful to the on-line users, it is important to make sure that the information obtainable is correct and up to date. One way to achieve this objective is to obtain as much information as possible from the system directly and automatically rather than to require that someone continually and manually update the information.

The list of system commands can be obtained from the command directory which is stored in core-A. Commands are either core-A transfer, core-B transfer, or core-b executable programs (saved files with a secondary name of TSSDC.). These latter types are stored in the system file directory (comfil 2 of M1416) and a check that the files exist in the directory can be made to verify that the command is in fact executable.

A list of the active supervisor entries can be made by examining the appropriate directory in core-A. The list of public programs can be obtained by examining the public file directory (comfil 4 of M1416, i.e., comfil P).

The subprograms which are available to be included in a user's program comprise a set of library files. Originally there was only one library, the TSSLIB file; but subsequently this file was broken down into TSLIB1 for general-purpose subprograms, TSLIB2 containing debugging programs, and KLULIB containing subprograms for the ESL display. To determine the programs contained within a library, a program can be written to read the program cards for each subprogram within the library file. The names of the entries to the subprogram, the transfer vectors or names of other programs which it calls, and the amount of core needed for loading (both relocatable and common) can thus be obtained. This information can be obtained by executing the command program PRBSS with a library file. Alternatively, the function performed by the PRBSS program can be incorporated within the on-line documentation system.

A library file can also be used together with a special program to produce a cross-reference table of the programs which are called by the entries in the library file. This information is needed when a change is being made to a program and it is necessary to reflect the change back

to the programs which call this program. This information has been prepared manually by J.Saltzer (Ref. 5) for the core-A subprograms which form the CTSS supervisor. The command program SUBUSE, prepared by B. Wolman at Project MAC, automatically prepares a reference table, of the type mentioned, by examining the program cards of subroutines included in a library file. By use of this program, an up-to-date reference table can be produced with no errors, which is unlikely when this job is done manually. The INFO system could be designed to accept a request to initiate this program, thus centralizing the information retrieval techniques.

Another way of insuring that up-to-date information is provided about a program is to require that an entry be made in the information system before a program can be added to the public file of programs or a new command added to the system. For example, the system could check to insure that for each core-B command (file with secondary name TSSDC.) a file exists with secondary name INFO. This could also be done for each public command in the public file directory, i.e., for each saved file. This technique is not completely satisfactory, but it indicates what could be done to coordinate the documentation of a program with the inclusion of the program in the system.

The difficulty with the above scheme is that (1)the size of the system and public file directories would be doubled by the inclusion of the INFO files, (2)there is no guarantee that the textual information provided by an INFO file is meaningful, and (3)this technique can't be used with the library subprograms which are combined into a single file, or for the core-A supervisor entries for which no files exist. The main problem centers around the problem of how the information is to be stored and how it is to be made accessible.

The INFO command is an information storage and retrieval system which has been designed in the context of the Compatible Time-Sharing System, and thus certain design decisions were based on the way auxiliary storage is handled within this environment. The general problem of storage and retrieval has not been considered, only that part of the general problem as it applies to the limited context of documenting the computer programs associated with CTSS. For example, certain information items were considered to be required in the documentation of a program, and this requirement was built into the STORE request. In a different context, information items other than those of this information system might be considered as required Also, particular features of the COMIT programming language are used to store the values of the items whose set of possible values are known. That is, in the present implementation, the information item TYPE is treated in a special way and can take on as values only COMMAND, ENTRY, LIBRARY, and PUBLIC. The context in which the INFO system is to operate has become an inherent part of its implementation (i.e., of the program).

The design of a truly general-purpose information storage and retrieval system would require that before it is used in a particular context, one specifies to the system the form that the information would take. For example, this specification could take the form of indicating the names of the required information items and the format in which they are to be interpreted and stored. A system to be used for bibliographic references might require the following information items:

(1) Title
(2) Author
(3) Publisher

(4) Date of publication

(5) Type, i.e., book, journal, report, etc.

Optional information items might be:

(6) References

(7) Page numbers

(8) Personal comments.

Once the specification of a system is made, it becomes a special-purpose system which is to be used in a particular context. A general-purpose information storage and retrieval system should be designed to operate in three modes. In the first mode, the specification of the format of the information base is made. This is done once, when the characteristics of the particular application are defined. The second mode is the storage mode where the information is provided to the system. The third mode is the retrieval mode. The second and third modes would both be available for operation during the use of the system. The procedures performed during the storage mode would make use of the specification of the information base but would not be dependent on any particular format of the information. Similarly, the types of retrieval processes that could be performed would be independent of the particular data base on which it was operating. A general-purpose information storage and retrieval system designed on these principles could be used in many different applications.

B. System Features

The INFO system has been designed to accept new information for storage or changes to information already stored according to information items. Verification by the on-line user is required whenever an attempt is made to remove the documentation of a program. A file containing information items for a given program may be created from the information stored in the system and, after it has been edited, this file, or one prepared with the use of the CTSS input facility, may be read by the INFO system to store or alter the information.

The user of the system can also list the values of certain information items which, in turn, may be useful in either the storage or retrieval process. Whenever a request results in a printout of over five lines, the system notifies the user of the number of lines which follow. The system can also perform a search for the program which satisfies particular conditions or has specific values for particular items. If the user makes an unsound request, the system balks and checks to see if the user really wants to make the request. If a search for programs satisfying particular conditions finds more than twenty matches, the system asks whether the user wants to continue the search.

The overall objective is to form a basis for obtaining textual information which describes a set of programs. This system could be combined with other special-purpose programs, such as a program which could automatically obtain the cross reference table of entries or calls, similiar to that prepared by Saltzer and Wolman. A considerable amount of information is obtained from the system itself, e.g., the list of active commands is obtained from the system command directory and the list of supervisor entries is obtained from the supervisor itself.

C. Language Features

A central philosophy of the man-machine communication language is that if a user is very familiar with the language and knows how the system behaves, he may communicate with it

in a very concise manner to accomplish his objectives. On the other hand, the novice, who is just learning the language may be very verbose and clumsy, but the language will lead him along, asking him questions and telling him what to do at every step of the way. If the user does not know what to do, he simply types a carriage return and the system will respond, telling the user what to do next. If the user has some experience and knows the format of information to be typed, he can be terse in his input statements.

The language used to communicate with the information system has been designed with the following principles in mind:

(1) The input format is independent of the request to the system.

(2) The format is semantic, rather than syntactic, thus making it simpler to learn, easier to understand, and more flexible in its use. For example, to specify that an item X has the value Y one may type the statement

THE VALUE OF ITEM X IS Y

rather than specifying just X and Y, where X is in one input field position and Y is in another input field position. The INFO system is permissive about the syntactic form of an input request, allowing words to be typed which may be ignored.

(3) The order of specifying items is not fixed, since item names must be supplied along with each information value.

(4) The system guides the user in steps indicating what to do. When an on-line user is in doubt of what to do, he merely types a carriage return.

(5) The user can anticipate input if he knows what is required, thereby reducing the on-line interaction.

(6) An experienced user can use abbreviations or eliminate redundant words, and hence simplify the on-line language.

D. Storage Considerations

The organization or data structure of the information to be stored is dependent on the types of retrieval to be performed. Trade-offs can be made between the ease of storing information and the ease of retrieving the information. In the documentation system, there are two types of information or data:

(1) Textual information which has no relation to other textual information and is retrieved by specifying the name of the body of textual data.

(2) Information which is cross related and on which various types of processing or searching are performed.

Each type of data should be kept in files separate from the program which processes the data. This storage organization allows the programs to be changed without affecting the data which it processes. Since the types of processing that are performed on these two types of data are different, the data structure should he different, and thus the information should be stored in separate files.

The textual information consists of groups of sentences to which a program name is associated. For each program name, there are generally more than ten lines of text. Since in the retrieval process only these lines are desired, it is not necessary to have all the textual information in core storage. Consequently, auxiliary mass storage in the form of random access files on the disk is used. In the present implementation, the body of textual information is stored

in three files (secondary names FILE1, FILE2, and FILE3) according to an equal partitioning of the set of first characters of a program name. If it is desirable to make a finer partition of program names, the first two characters of a name may be used.

Each file consists of an integral number of tracks on the disk and, for efficiency of storage, the information should be stored in such a way as to minimize the unused storage space on a track. Assume that, on the average, each file contains one track which is only half used. The more information stored in a file, the smaller is the percentage of wasted storage space. But, if a file contains the textual information for more than one program, this file must be searched linearly to obtain the text of a desired program. If more files are used, then, on the average, the linear search for textual information is shortened, since each FILE would contain less information.

On the other hand, when the body of textual information is stored in many shorter files rather than in fewer larger files, the percentage of wasted storage is increased and the available disk storage space is used less efficiently. In addition, when more files are used, the CTSS supervisor is burdened with keeping track of the names of each file and its location on the disk. Thus, the trade-offs between efficiency of storage and ease of retrieval should help to determine the optimum number of files to use.

The organization of the search information depends on the types of retrieval to be performed. If the search data are cross related in such a way that various associations can be made between the items of data, it is desirable to store each item of data only once and use pointers to indicate the relations among the data. List structure techniques, where one list can be a sublist of many lists, can be useful in implementing these relations. Other advantages of list structures are:

(1) The number of words or entries with which the program has to deal does not have to be predicted in advance, thus eliminating the necessity of reserving fixed length blocks of storage.

(2) Storage space once used can be put back on a free storage list when it is no longer needed, thus making it available again when it is needed.

(3) The program is relieved of the problem of allocating a fixed storage location for the data, since the list of available space links together the usable storage space.

The search information in the INFO system consists of the information items – TYPE, AUTHOR, DATA, LANGUAGE, and DESCRIPTOR which are associated with a program NAME. This search information is stored in core by using the list or string structure of the COMIT language (Sec. VI). The data for a given program are stored as two constituents plus one constituent for each DESCRIPTOR. The first constituent consists of the program NAME with the values for the information item program TYPE as subscript values to the logical subscript TYPE. The second constituent consists of an AUTHOR value with the DATE documented as its numerical subscript and the LANGUAGE value as the subscript value to the logical subscript LANG. Each DESCRIPTOR is stored as a single constituent following these first two, and the data for each program are separated by a constituent with the special symbol *X. The data are stored on 47 SHELVES (linear strings in COMIT) corresponding to the 47 different possible first characters of a program name. A finer or coarser partition could be made by a simple change in the program. The more shelves that are used, the easier it is to obtain the data of any given program since, on the average, the amount of data on any given shelf is reduced.

In the present implementation of the INFO system, the search data are stored in one file (with second name DATA), and when these data are loaded into core they are stored on the

3-28-6425

```
-STORAGE *= = SHELF/.80 + --DATA + -FILE1 *
    -  = SHELF/.81  + --DATA + -FILE1 *
    *+ = SHELF/.82  + --DATA + -FILE1 *
    .  = SHELF/.83  + --DATA + -FILE1 *
    *1 = SHELF/.84  + --DATA + -FILE1 *
    *2 = SHELF/.85  + --DATA + -FILE1 *
    *3 = SHELF/.86  + --DATA + -FILE1 *
    A  = SHELF/.87  + --DATA + -FILE1 *
    B  = SHELF/.88  + --DATA + -FILE1 *
    C  = SHELF/.89  + --DATA + -FILE1 *
    D  = SHELF/.90  + --DATA + -FILE1 *
    E  = SHELF/.91  + --DATA + -FILE1 *
    F  = SHELF/.92  + --DATA + -FILE1 *
    G  = SHELF/.93  + --DATA + -FILE1 *
    H  = SHELF/.94  + --DATA + -FILE1 *
    *) = SHELF/.95  + --DATA + -FILE2 *
    *- = SHELF/.96  + --DATA + -FILE2 *
    *$ = SHELF/.97  + --DATA + -FILE2 *
    ** = SHELF/.98  + --DATA + -FILE2 *
    *4 = SHELF/.99  + --DATA + -FILE2 *
    *5 = SHELF/.100 + --DATA + -FILE2 *
    *6 = SHELF/.101 + --DATA + -FILE2 *
    I  = SHELF/.102 + --DATA + -FILE2 *
    J  = SHELF/.103 + --DATA + -FILE2 *
    K  = SHELF/.104 + --DATA + -FILE2 *
    L  = SHELF/.105 + --DATA + -FILE2 *
    M  = SHELF/.106 + --DATA + -FILE2 *
    N  = SHELF/.107 + --DATA + -FILE2 *
    O  = SHELF/.108 + --DATA + -FILE2 *
    P  = SHELF/.109 + --DATA + -FILE2 *
    */ = SHELF/.110 + --DATA + -FILE3 *
    ,  = SHELF/.111 + --DATA + -FILE3 *
    *( = SHELF/.112 + --DATA + -FILE3 *
    *7 = SHELF/.113 + --DATA + -FILE3 *
    *8 = SHELF/.114 + --DATA + -FILE3 *
    *9 = SHELF/.115 + --DATA + -FILE3 *
    *0 = SHELF/.116 + --DATA + -FILE3 *
    Q  = SHELF/.117 + --DATA + -FILE3 *
    R  = SHELF/.118 + --DATA + -FILE3 *
    S  = SHELF/.119 + --DATA + -FILE3 *
    T  = SHELF/.120 + --DATA + -FILE3 *
    U  = SHELF/.121 + --DATA + -FILE3 *
    V  = SHELF/.122 + --DATA + -FILE3 *
    W  = SHELF/.123 + --DATA + -FILE3 *
    X  = SHELF/.124 + --DATA + -FILE3 *
    Y  = SHELF/.125 + --DATA + -FILE3 *
    Z  = SHELF/.126 + --DATA + -FILE3 *
```

Fig. 1. File Specification LIST.

47 shelves. If more DATA files were used, by partitioning the search data in a different way, the amount of data on a given shelf from a given DATA file would be reduced. If all the DATA did not have to be searched, this would result in a reduction in the amount of data that must be read and loaded into core storage. Since quite often it is necessary to search all the data, no savings would be obtained in this case. The amount of search data that can be stored in core at one time is limited and, as this amount grows, the system is eventually forced to split the data into more than one DATA file.

In the present system, the search data use approximately fifteen words per entry. The system has 10,000 words of available free storage, which is used to store the INFO DATA file and to process the textual information for a single program. About 1000 words of free storage should be reserved for processing requests and textual information of a single program. This leaves room for search data for about 600 programs. When more space is required, the INFO DATA file can be partitioned into multiple sections. The present partitioning of the information for storing in files is illustrated in the File Specification List (a COMIT list rule) shown in Fig. 1.

E. Console Printing

Information on the documentation of a program is conveyed to a user through an on-line console. The primary types of consoles presently in use with CTSS are (1) the Model 35 Teletype, (2) the IBM 1050 Selectric, and (3) the ESL display. Each type of console has a different set of characters associated with it, and sends and receives different character signals. The CTSS supervisor performs all the code conversions for transmitting and receiving characters between a remote console and a program within the computer. The CTSS supervisor, by rules of convention, maps each character signal received into a BCD code for representation within the computer and maps codes generated by a program into signals for transmission to a console for printing.

Characters are represented in the computer in one of two modes; in the "normal" mode, i.e., as a 6-bit code, or in the "full" mode, i.e., as a 12-bit code. In the 12-bit mode, the high-order 6-bits are referred to as logical case bits. Some characters can only be represented in the "full" (12-bit) mode, and others may also be represented in the "normal" (6-bit) mode. These mappings between signals and codes are peformed by the supervisor in one of the two modes which is set by program control.

If the mapping is performed in the "normal" mode, and a character which can only be represented in the "full" mode is typed, the character may be converted to the corresponding "normal" mode character by deleting the case bits, or it may be ignored. The characters which are convertible depend on the console being used and are usually restricted to the set of lower case letters (represented in the 12-bit mode only) which are converted to upper case letters (represented in the 6-bit mode). Thus, when operating in the "normal" mode a user may type a character which is ignored, or he may type a character which gets converted to a different character. If an attempt is made to print a character which does not exist on a particular console, the character is either converted to a printable character or is ignored.

The INFO system has been implemented for use in the "normal" mode, and hence only the "normal" BCD set of characters may be printed (although some of the characters in the "full" set may be typed and converted into characters of the "normal" BCD set). Consequently, textual

descriptions of a program are printed in all upper case letters regardless of the console being used. This makes reading and comprehending of the information slightly more difficult.

If the INFO system were designed to operate in the "full" mode, both upper and lower case letters could be printed on those consoles which have them. For those consoles which do not have lower case letters, the CTSS supervisor would map them into upper case letters. The problem with this mode of operation is that twice as much storage space would be required to store the textual information; consequently, the average search time for the documentation of a program would be doubled.

It is important to be able to program a time-shared computer for on-line interaction in such a way that it is not dependent on the console being used. Each character should be given a unique representation within the computer, regardless of the console being used. In this way, programs could be written with the assumption that the characters output to a console for printing will be the same, regardless of console. This is not the case in the present design of the CTSS adapter module. In addition, it is useful to have two modes of operation, one in which characters output are represented uniquely (the "full" set) and a second in which some characters are converted to "normal" set characters.

F. System Response

A user sitting at his console makes a request to the system and desires the system to respond immediately, i.e., within the human reaction time, which is of the order of 2/10 of a second. The system's response is governed by the amount of time required to process a user's request and the amount of time which is necessary for the system to communicate a complete response to the user.

The real time required to process a request depends on the complexity of the request, i.e., the amount of processing that has to be done, and the scheduling algorithm which determines what portion of real time is allotted to a particular user for his computation. A user does not have control over the algorithm used to determine his priority of service; hence, he can only attempt to minimize his demand for computation which is based on the complexity of processing that is required. It will be indicated that the amount of time required for a particular process is based on the structure of the data or information stored.

The time required for the system to communicate with a user depends on the nature of the on-line console. The output from an on-line console may either be printed (by a typewriter or a plotter), displayed (as lights or as a picture), or punched (on punched paper tape or on punched cards). The time required for each type of output is different, and this correspondingly influences the information system's response to the on-line user. In general, the user does not consider that the system has responded until the output is completed and it is the user's turn to act. Thus, if a user makes a request requiring that ten lines of output be printed, he waits until all ten lines have been printed before he begins to read the lines to complete the system-user portion of the communications cycle. The time required for a page of text to be displayed on a viewing device is at least an order of magnitude faster than the time it takes for the page to be printed on a typewriter. Consequently, in discussing a system response to a user's request, one should keep in mind the response time of a particular on-line console.

The typewriter console used with the present CTSS system imposes a considerable delay when more than ten lines of output are produced during one man-machine cycle. Because

of this fact, it is important that the on-line user have the ability to request from the system only the information which he wants to have, without producing additional unwanted information which consequently increases the response time of the system and decreases the percentage of useful information. Even a slight difference in console communication speed is noticeable to the on-line user. Users find the IBM 1050 console preferable to the teletype console primarily because of the former's faster typing speed. It appears that this is more important than the difference in console key layout, since a user can easily adapt to different key positions.

If display devices for textual information were more accessible, this consideration would not be so crucial to the user-system response. A page of textual information can be displayed much faster than it can be typed, tremendously improving the communications between the system and the on-line user. Several techniques are suggested for displaying continuous pages of text.

Consider that a page is made up of a fixed number of lines, say twenty, depending on the resolution of the display device. Successive lines are displayed on a page as they are generated, until the page is full. A full page may be indicated either by a light or by a marker displayed at the bottom of the page. When the user has viewed the page, he might push a button, flick a switch, type a carriage return, or make an indication with a light pen to view the following page.

Alternatively, as one page becomes filled, the top few lines could be made to disappear and the rest of the page moved up so that additional lines could be displayed on the bottom. The amount of the page that is moved up could be controlled by a continuous knob which the user could turn as if he were rolling up a scroll. With this technique, the user always has displayed in front of him a portion of a previous page, i.e., a number of lines preceding the last line displayed. The importance of many of these considerations to the on-line operation of a computer should influence the design of future time-shared computers.

VI. THE PROGRAMMING LANGUAGE

A. The COMIT Language

The INFO command has been written in the COMIT programming language (Ref. 6) which has recently been adapted for use with CTSS. COMIT was chosen because it is well suited for string manipulation of textual material. It uses a linked string structure for storage of data; hence, no limit is imposed on the length of the English words or the nature of the text. A COMIT program is easy to modify during the trial and error procedures of developing a suitable communication language between man and machine, i.e., between the user and the on-line program. The built-in string manipulation and searching features of COMIT permit easy experimentation of processing algorithms, for example, in defining a new search routine. The version of COMIT used has some of the new COMIT II features, in particular, the ability to execute a binary subroutine (assembled in FAP or MAD) upon transfer from the "go-to" of a COMIT rule.

The data which are manipulated or processed by a COMIT program consist of constituents to which may be associated one numerical subscript and any number of logical subscripts which may take on up to 36 logical values. A constituent is a concatenation of any number of BCD characters; whereas, logical subscripts and subscript values are a concatenation of up to 12 BCD characters. Constituents are connected, through the use of pointers, in a linear string. There may be 128 strings of constituents which are referred to as the WORKSPACE and the 127 SHELVES.

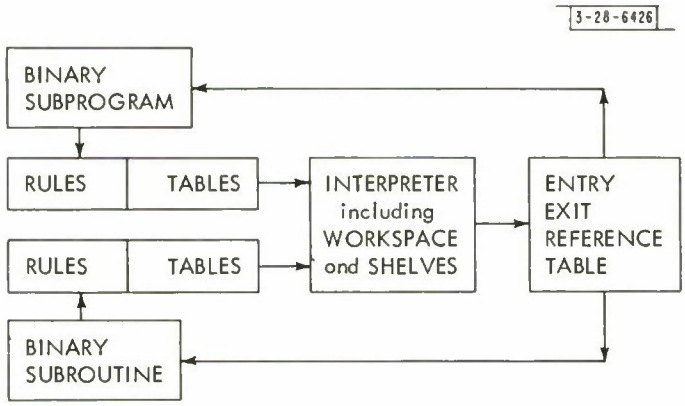

Fig. 2. Paths of control between binary subprograms and COMIT routines.

3-28-6427

```
PROG1     SXA     RETURN, 4
          TSX     $.COMIT, 4
          TXH     RULES
          TXH     RULENO
RETURN    AXT     **, 4
          TRA     1, 4
RULES     PZE     TABLES
            .
            .
            .
TABLES
            .
            .
            .
```

Fig. 3. Form of compiled COMIT program.

A COMIT program consists of a set of rules which are executed interpretively. A rule consists of the following: a rule name and optional subrule names; a left-half for matching with the constituents in the WORKSPACE; a right-half for specifying the manipulation of constituents found in the left-half; a routing for indicating operations to be performed with the WORKSPACE constituents, with the SHELVES, or with the input/output devices; and a go-to to specify which rule is to be interpreted next. A go-to may specify the execution of a binary subroutine rather than a COMIT rule.

The running of a COMIT program consists of compiling the rules into a compact coded form and producing reference tables of rule names, subrule names, subscript names, and subscript values. The compiled program is then interpreted with reference to the associated tables and with a possible transfer to binary subroutines compiled by the FAP or MAD translator. In the present version of COMIT, the binary subroutines must be loaded into core together with the COMIT compiler and interpreter. At the time of execution, the core space used by the compiler is available as free storage for inclusion in the WORKSPACE.

Improvements in the organization of the COMIT system are being made which will make COMIT more usable as a programming language. For example, the present version of COMIT requires that all the available core storage be assigned to a COMIT program, whether or not all the storage is needed. The COMIT system is being modified so that only the portion of storage which is required at any given moment is assigned to the program. Thus, the amount of storage assigned can change dynamically as the requirements of a running program vary. In particular, the amount of storage needed for the INFO system multiplies with an increase in search data. The amount of core storage presently required for the INFO system is about 20,000 registers; thus, with the modification to the COMIT system, a two-thirds savings would be obtained. Consequently, the amount of time required to load and swap the command would be reduced. When this improvement is made, the INFO system can be recompiled to take advantage of the savings.

To facilitate the use of COMIT with programs compiled by other translators, and in particular for use in CTSS, the following modification to the COMIT system is proposed.

In order to enable binary subroutines to use a subroutine written in COMIT, the COMIT compiler should write a binary file containing the coded COMIT rules and its associated tables in the form of relocatable binary card images preceded by an entry sequence. The COMIT interpreter would be split from the compiler and added to the CTSS BSS library file. When the binary routines are loaded into core with a BSS loader (one of the standard CTSS LOAD commands) the compiled COMIT program is also loaded together with the Entry Exit Reference Table (a FAP program through which COMIT calls binary subroutines) and the COMIT interpreter (obtained from the library). Figure 2 indicates the paths of control between COMIT routines and other binary subprograms which may be loaded into core at the same time.

The COMIT interpreter contains all the machinery for interpreting rules, manipulating the WORKSPACE and SHELVES, and allocating storage from its storage list. When a routine calls a COMIT subroutine, the "entry sequence" stores index register four in order to return to the calling program and transfers to the COMIT interpreter with, as parameters, the beginning location of the rules, and the number of the rule which is to be interpreted first (the rules are assigned sequential numbers at compilation time).

The compiled COMIT program together with the entry sequence might take the form as shown in Fig. 3, where .COMIT is the entry to the COMIT interpreter.

Note that the location of the TABLES is given indirectly from RULES.

This scheme can easily be incorporated with the changes now being made to the COMIT system. Besides the modifications already indicated, the COMIT language must be adapted to permit the specification of an entry point at a particular COMIT rule, and for each entry the compiler must generate an entry sequence with an appropriate value for the parameter RULENO. If no entry is specified, the compiler must generate an entry sequence with RULENO set to the first rule of the program, the return set to COMEND (the standard procedure for terminating a COMIT program), and with the initial instruction for the storing of the return (index 4) eliminated. Also, the COMIT interpreter must be modified to return to the calling subroutine when an END rule is encountered.

The procedure outlined above is not recursive; once one COMIT routine is entered it cannot be entered again until it is completed, i.e., it falls to the END rule. It may call a binary routine or another COMIT routine, but these other routines cannot call the original COMIT routine. It does enable large COMIT programs to be written in pieces and permits the features of COMIT to be used together with the features of other languages.

B. Use of COMIT Features

Certain features of COMIT have been most appropriate for programming the INFO system. These are:

(1) The automatic handling of available core storage space by means of a free storage list.

(2) The automatic storing, via string pointers, of any number of characters as a constituent, i.e., for a single word or for the complete textual description of an information item.

(3) The list structuring of the SHELVES which is used for partitioning the data alphabetically and for facilitating access or addressing of portions of the data.

(4) The left-half searching of the data for information items with matching DESCRIPTORS, TYPE, LANGUAGE, and AUTHOR, or for a DATE (stored as a numerical subscript) which is greater than a given value.

(5) The right-half specification of output format for printing on a console or for storing data in a file.

(6) The simple input/output routing conventions for reading lines typed on a console or for printing lines on the console.

VII. ADDITIONAL MODIFICATIONS

The design of the INFO system has progressed through many stages of modification. In an early stage of the design, the request language was awkward and required a fixed and stylized format. Experience in an on-line environment led to improvements in the request language until the present form appeared satisfactory. In the present system, requests can be typed to the INFO system or, alternatively, the request to describe the documentation of a specified program can be initiated by specifying the program NAME as a command parameter when the INFO command is called. A further modification to the system would permit all requests to be specified as command parameters. Thus, one would be able to resume the INFO command and specify a request in one line of type. For example

INFO D NAME IS TYPSET, USAGE AND AUTHOR

to initiate a describe request. The present design of the CTSS supervisor limits the number of command parameters to twenty, and each parameter must be six characters or less. There

would be no problem specifying a program NAME since a name is at most six characters, but an AUTHOR or DESCRIPTOR might be more than six characters and, hence, could not be given as a command parameter. Also, there would be no problem specifying requests or item names, since these can be abbreviated. In a future design of a time-sharing system on a different computer, these limitations can be eliminated.

As was discussed in an earlier section, it would be useful to define sub and super categories of descriptors. The implementation of a hierarchy of descriptors could be incorporated into the present descriptor list by tagging each descriptor with a level indicator. A modification could be made to the LIST request so that a request to LIST the DESCRIPTORS would give only the descriptors which are in the subcategory of a specified descriptor. Further study could be made to determine a good way to define descriptor categories.

The classification of programs has been done superficially in the SHARE index of distributed programs (Ref. 7), but these are not completely satisfactory. Broad categories such as arithmetic, input/output, code conversion, etc., are only of limited use. More specialized categories are needed and other associations must be made among programs which can be used to help locate or pick out a program satisfying a particular need.

The problem of determining the programs which are of interest cannot be solved by category retrieval alone. More sophisticated techniques are needed. For example, a dictionary of synonyms and antonyms might be useful for expanding the descriptor language and adapting it to different contexts. A dictionary of related words could be used to define additional relations among programs. If a dictionary entry related the descriptors INPUT and EDIT, a request for documentation with descriptor EDIT would also provide the information which had the descriptor INPUT. Techniques for analyzing English sentences might be useful in determining the nature of a request and providing the necessary semantic information. More research is needed to devise techniques of category retrieval.

As a future modification to the INFO system, the portion of the program which is concerned with the STORAGE requests could be separated from that portion which is concerned with the RETRIEVAL requests. As a result, the size of each portion of the program would be reduced. More core storage would thus be available for search data during retrieval processing. Or, alternatively, when the size of the retrieval program is reduced, the program load and swap time are decreased, resulting in an improvement in system response. The present model of on-line documentation system was implemented as a single system to make use of common processing procedures and to coordinate the programming effort.

The allocation of work to develop a programming system is divided among (a) the writing of the source program, (b) the compiler or translator, and (c) the functions performed by a supervisor or monitor system. In the design of the present INFO system, the on-line communication language was specified in the source program; the COMIT compiler was used to handle storage allocation and data structure; and the CTSS supervisor controlled message communications between the program and the on-line users, and the storage of information on disk files. Each of these features might be better implemented by a redesign of the system, as is true when developing new programming techniques.

VIII. SUMMARY

The on-line documentation system, as implemented, serves as a model for demonstrating the usefulness of an on-line documentation system and for designing the suitable language for

communicating with the system. The system indicates some useful tools for better documentation of computer programs in an on-line environment. It is hoped that through use of this system, insight into the area of program documentation can be obtained.

On-line users of a computation facility demand more up-to-date documentation of the available programs, and an attempt to satisfy their demand should be made. Also, the problem of communication is greatly increased when users of the computation facility are remote from the computer and the administrative staff. As the computer becomes more like a utility, adequate communications of all types and in all forms must be provided.

REFERENCES

1. F. J. Carbató, et al., The Compatible Time-Sharing System: A Programmer's Guide (M. I. T. Press, Cambridge, Massachusetts, 1963).
2. M. M. Janes, E. C. Van Horn, and J. M. Winett, "A System for Storing and Retrieving Information about Computer Programs," Course 6.543 Seminar Paper, M. I. T., Cambridge, Massachusetts (16 December 1963).
3. Computation Center Memoranda, Project MAC Memoranda, Time-Sharing Technical Notes, CTSS Bulletins, and Programming Staff Notes.
4. U. Neisser, "MAC and its Users," Project MAC Memorandum MAC-M-185, M. I. T., Cambridge, Massachusetts (29 September 1964).
5. J. H. Saltzer, "CTSS Technical Notes," Project MAC Technical Report MAC-TR-16, M. I. T., Cambridge, Massachusetts (1965).
6. V. Yngve, COMIT Programmer's Reference Manual (M. I. T. Press, Cambridge, Massachusetts, 1961).
7. SHARE Index of Distributed Programs, M. I. T. Computation Center.

APPENDIX A

PRINTING FROM A SESSION WITH THE ON-LINE INFO SYSTEM

(Words in lower case were typed by an on-line user.)
(Words in upper case were typed by the INFO system.)
(This session with the INFO system was on 1/07/65.)

resume info
W 403,6
TYPE REQUEST, OR C.R. FOR INSTRUCTIONS..

list the items
REQUIRED ITEMS ARE – NAME, TYPE, DESCRIPTORS, DATA, AUTHOR, LANGUAGE, PURPOSE, USAGE, AND REFERENCE.
OPTIONAL ITEMS ARE – SIZE, AND TV.
OK..

list authors and then list discriptors
AUTHORS ARE – .STAFF CAMPBELL CORBATO DAGGETT POUZIN JONES MINSKY ROSS WINETT YNGVE
OK..

DESCRIPTORS ARE – . BOOLEAN COMMANDS COMPILER CONVERSION CONSOLE DEBUG DOCUMENTATION EDITING ERRORS EXECUTION EXITS FILE FMS DEBUG DOCUMENTATION EDITING ERRORS EXECUTION EXITS FILE HARDWARE I/O LOADER LOG MISC. PROGRAM READING SIMULATION STATUS SUPERVISOR TAPE TEST TIMER TRANSLATOR UTILITY WRITING
OK..

list the programs whose type is command

SYSTEM COMMANDS

LOGOUT	LOGIN	ENDLOG	INPUT	EDIT	START
OCTPAT	OCTTRA	LOAD	NCLOAD	VLOAD	LOADGO
PM	STOPAT	TRA	PATCH	FAPDBG	STRACE
LISTF	PRINTF	FILE	TFILE	SAVE	RESTOR
R	MAD	MADTRN	CHMODE	DELETE	RENAME
SPLIT	SD	SP	COPY	UPDATE	COMFIL
AED	COMIT	LISP	ED	SNOBOL	PRBSS
EXTBSS	UPDBSS	RUNCOM	SDUMP	GENCOM	LDABS
MEMO	MODIFY	DITTO	REMARK	DYNAMO	RQUEST
ARCHIV	BEFAP	CRUNCH	LOG	BLODI	STRESS
COGO	TYPSET	RUNOFF	CTEST1	CTEST2	CTEST3
CTEST5	CTEST6	CTEST7	CTEST8	CTEST9	CTEST4
OCTLK	USE	FAP	RESUME	COMBIN	GPSS
PRBIN	OPL	PRINT	MADBUG		

OK..

list type is entry

SUPERVISOR ENTRIES

WRFLX	WRFLXA	RDFLXA	.WRITE	.DUMP	.LOAD	.READK
.ASIGN	.APEND	.RELRW	.SEEK	.FILE	.ENDRD	.CLEAR
.FSTAT	.DLETE	.RENAM	.RESET	.UPDAT	.FILDR	.GTFLG
INSTRT	INPEND	GETMEM	GETCOM	SETMEM	(LFTM)	(EFTM)
DEAD	DORMNT	FNRTN	NEXCOM	GETILC	TRA1,4	RSTIME
DEFERR	AKNOLG	TSSFIL	USRFIL	SETFUL	SETBCD	SETBRK
SAVBRK	GETBRK	SETCLC	GETCLC	GETCLS	SETCLS	CHNCOM
SETLOC	GETLOC	COMFIL	SETMON	SETUSR	.ERASE	EREAD
DSCOPE	GETMON	GETIME	LOGINA	CLOCON	CLOCOF	SLEEP
SELECT	CHECK	MONSCD	MONINF	PCTSS	RDMESS	WRMESS
FORBID	ATTCON	REDLIN	RELEAS	ALLOW	SET6	SET12
SNDLIN	SNDLINA	SLAVE				

OK..

```
desc n is ed, da pu
    DATE IS – 1/14/65
    PURPOSE IS – FOR INPUTING OR EDITING 14 WORD BCD CARD IMAGE FILES
    USING CONTEXT EDITING.

OK..

find des is utility, a is staff
20 ITEMS FOUND SO FAR.  DO YOU WANT TO CONTINUE..
yes

ARCHIV, BLODI, CTEST7, CTEST6, CTEST5, CTEST1, COPY, COMFIL, COMBIN,
CHMODE, DELETE, EXTBSS, EDIT, FILE, GENCOM, INPUT, LISTF, LOG,
PRBIN, PRBSS,

32 ITEMS FOUND.

PRINT, PRINTF, R, RQUEST, REMARK, RUNCOM, RESUME, RESTOR,
SPLIT, SAVE, UPDBSS, UPDATE,

OK..

describe name is prbss, usage
USAGE IS – (6 LINES FOLLOW)
PRBSS 'A' 'B'
LIBRARY FILE 'A' BSS IS READ AND A SUMMARY OF ALL BSS PROGRAMS
BEGINNING WITH THE PROGRAM WITH ENTRY NAME 'B' IS PRODUCED.
IF 'B' IS OMITTED THE SUMMARY BEGINS AT THE BEGINNING.

OK..

f l is mad
5 ITEMS FOUND.

ED, MADBUG, PRBIN, PRBSS, PRINT,

OK..

f date is 1/01/65
20 ITEMS FOUND SO FAR.  DO YOU WANT TO CONTINUE..
yes

AED, ARCHIV, BLODI, BEFAP, CTEST9, CTEST8, CTEST7, CTEST6,
CTEST5, CTEST4, CTEST3, CTEST2, CTEST1, COPY, COMIT,
COMFIL, COMBIN, CHMODE, COGO, CRUNCH,

40 ITEMS FOUND SO FAR.  DO YOU WANT TO CONTINUE..
yes

DITTO, DELETE, DYNAMO, ED, ENDLOG, EXTBSS, EDIT, FILE,
FAPDBG, FAP, GPSS, GENCOM, INPUT, INFO, LOGOUT, LOGIN,
LOADGO, LISTF, LISP, LDABS,

60 ITEMS FOUND SO FAR.  DO YOU WANT TO CONTINUE..
yes

LOG, LOAD, MADBUG, MADTRN, MAD, NCLOAD, OPL, OCTTRA,
OCTPAT, OCTLK, PRBIN, PRBSS, PRINT, PRINTF, PATCH, PM,
R, RQUEST, REMARK, RUNCOM,

78 ITEMS FOUND.

RENAME, RESUME, RESTOR, STRESS, STOPAT, SDUMP, SNOBOL,
SP, SD, SPLIT, SAVE, START, TFILE, TRA, UPDBSS, UPDATE,
USE, VLOAD,

OK..

d n is ditto, r
REFERENCE IS – GUIDE PP. 82-86
SYSTEM NOTES NUMBER 3
BULLETIN 5 AND 32.

OK..

d n is typset
TYPSET NOT DOCUMENTED.
OK..
```

```
store file is typset
FILE TYPSET INFO BEING READ.

FILE TYPSET INFO BEING READ TO OBTAIN ITEMS FOR TYPSET PROGRAM.
SALTZER IS A NEW AUTHOR, CORRECT IT OR TYPE OK OR IGNORE..
ok

REFERENCE IS..
memo mac-193 cc-244
OK..

d n is typset
NAME IS - TYPSET
TYPES ARE - COMMAND
DESCRIPTORS ARE - UTILITY EDITING
DATE IS - 1/15/65
AUTHOR IS - SALTZER
LANGUAGE IS - MAD
PURPOSE IS - USED TO INPUT AND EDIT 12-BIT (FULL MODE) MEMO FILES.
EDITING IS BY CONTEXT AND WITHOUT LINE NUMBERS.
OFTEN USED WITH RUNOFF TO PREPARE MEMOS.

USAGE IS - (30 LINES FOLLOW)
TYPSET 'NAME'
'NAME' IS FIRST NAME OF FILE (IF FOUND) WITH SECOND NAME (MEMO).
THERE ARE TWO MODES OF OPERATION, INPUT AND EDIT.
IN INPUT MODE LINES MAY BE TYPED CONTINUOUSLY
WITHOUT RESPONSE FROM THE TYPSET COMMAND.
TO CHANGE MODES TYPE A C.R.
REQUESTS MAY BE ABBREVIATED BY THEIR FIRST LETTER.
IN EDIT MODE REQUESTS ARE..
    FIND 'LINE'
TO FIND LINE BEGINNING WITH THE NON BLANK CHARACTERS IN 'LINE'
    LOCATE 'STRING'
TO FIND THE LINE CONTAINING THE 'STRING' BEGINNING IN ANY COLUMN
    NEXT 'I'
TO MOVE TO THE NEXT I-TH LINE
    DELETE 'I'
TO DELETE THE NEXT I LINES INCLUDING THE PRESENT ONE.
    PRINT 'I'
TO PRINT I LINES.
    RETYPE 'LINE'
THE PRESENT LINE IS REPLACED WITH 'LINE'
    TOP
THE CURRENT POINTER IS SET TO BEFORE THE FIRST LINE IN THE FILE
    BOTTOM
INPUT MODE IS ENTERED TO ADD LINES AT THE END OF THE FILE
    INSERT 'LINE'
THE 'LINE' IS INSERTED AFTER THE CURRENT LINE
    CHANGE *STRING1*STRING2* I G
STRING2 IS MADE TO REPLACE STRING1 IN I LINES
IF G IS GIVEN ALL OCCURRENCES OF STRING1 IN A LINE ARE REPLACED.
'*' MAY BE ANY CHARACTER.
    VERIFY ON/OFF
IF ON - FIND, NEXT, LOCATE, AND CHANGE REQUESTS WILL BE VERIFIED
IF OFF - NO VERIFICATION WILL BE MADE.
    SPLIT 'NAME'
THE LINES BEFORE HERE ARE FILED WITH NAME 'NAME'
    ERASE 'X'
'X' IS SET TO THE ERASE CHARACTER
    KILL 'X'
'X' IS SET TO THE KILL CHARACTER

REFERENCE IS - MEMO MAC-193 CC-244

OK..

(  The INFO command is described by the on-line system as    )
(  follows.  This description can be obtained by typing a    )
(  C.R. after the INFO system is resumed.                    )
```

```
d n is info
NAME IS – INFO
TYPES ARE – COMMAND
DESCRIPTORS ARE – DOCUMENTATION
DATE IS – 1/08/65
AUTHOR IS – WINETT
LANGUAGE IS – COMIT
PURPOSE IS – (22 LINES FOLLOW)
    THIS IS AN ON-LINE SYSTEM FOR STORING AND RETRIEVING INFORMATION
ABOUT THE FOLLOWING TYPES OF PROGRAMS ASSOCIATED WITH CTSS –
SYSTEM COMMANDS, SUPERVISOR ENTRIES, LIBRARY SUBPROGRAMS, AND
PUBLIC PROGRAMS.
    THE FOLLOWING ITEMS OF INFORMATION ARE AVAILABLE ABOUT A
PROGRAM –
    PROGRAM NAME (N) – A SINGLE WORD.
    PROGRAM TYPE (T) – ONE OF THE FOLLOWING.. COMMAND, ENTRY,
        LIBRARY, AND PUBLIC.
    DESCRIPTORS (D) – KEY WORDS USED TO CLASSIFY THE PROGRAMS
        IN THE INFORMATION FILES.
    PURPOSE (P) – A SHORT ABSTRACT INDICATING THE CONTEXT IN
        WHICH A PROGRAM MIGHT BE USED.
    USAGE (U) – THE INSTRUCTIONS OF HOW TO USE THE PROGRAM.
    PROGRAMMING LANGUAGE (L) – THE LANGUAGE IN WHICH THE PROGRAM
        IS WRITTEN.
    REFERENCE (R) – A BIBLOIOGRAPHY OF WHERE MORE INFORMATION
        ABOUT A PROGRAM MAY BE OBTAINED.
    AUTHOR (A) – THE NAME OF THE PERSON WHO IS RESPONSIBLE
        FOR THE PROGRAM.
    DATE (DATE) – THE DATE THE INFORMATION WAS LAST STORED OR
        ALTERED.
USAGE IS – (23 LINES FOLLOW)
    TO USE THIS INFORMATION SYSTEM, TYPE THE COMMAND 'INFO'.
REQUESTS TO THE SYSTEM ARE DESCRIBE (D), LIST (L), FIND (F),
STORE (S), ALTER (A), EDIT (E), AND REMOVE (R). TO OBTAIN A
DESCRIPTION OF EACH REQUEST, TYPE THE REQUEST NAME ONLY.
    REQUESTS TO THE SYSTEM SPECIFY AN ITEM NAME, OR AN ITEM
NAME TOGETHER WITH AN ITEM VALUE ASSOCIATED WITH THE ITEM NAME
IN THE FORM 'ITEM NAME' IS/ARE 'ITEM VALUE'. THE REQUEST NAME IS
TYPED FIRST FOLLOWED BY ITEM NAMES AND ITEM VALUES WHEN
APPROPRIATE. AN ITEM VALUE BEGINS WITH THE WORD 'IS' OR 'ARE'
AND MUST END WITH A COMMA. THE WORD 'THEN' INDICATES THE TERMINATION
OF A REQUEST AND THUS ALLOWS MULTIPLE REQUESTS TO BE TYPED.
WORDS OTHER THAN ITEM NAMES OR ITEM VALUES OR THE WORD 'THEN' MAY BE
TYPED BUT ARE IGNORED.
    REQUESTS AND ITEM NAMES MAY BE ABBREVIATED BY THEIR FIRST LETTER.
IF THE WORD 'QUIT' IS TYPED IN A FIELD IN PLACE OF AN ITEM NAME
THE PRESENT REQUEST IS IGNORED. IF TWO OR MORE INPUT FIELDS SPECIFY
THE VALUE OF AN ITEM THE LAST VALUE TYPED TAKES PRECEDENCE.
    TO CONTINUE INPUT ON ANOTHER LINE PRECEDE THE CARRIAGE RETURN
(C.R.) BY A DASH (–). TO INCLUDE A COMMA (,) AS TEXT IN AN ITEM,
FOR EXAMPLE IN A SENTENCE DESCRIPTION, PRECEDE THE COMMA BY A
STAR (I.E.*,). TO INCLUDE A C.R. AS TEXT IN AN ITEM PRECEDE
THE C.R. BY A STAR (*).
    WHEN IN DOUBT OF WHAT TO DO, TYPE A C.R.
REFERENCE IS – SEE JOEL WINETT, EXT 6039 OR 81-301.
OK..
(   Each request can be described by typing the request    )
(   name only.                                            )
list
    LIST I(1) . . . I(N), TYPE IS 'TYPE'
    THE LIST REQUEST IS USED TO OBTAIN A LIST OF THE NAMES OF ALL
INFORMATION ITEMS, THE VALUES OF CERTAIN INFORMATION ITEMS, OR
OF A PARTICULAR TYPE. THE REQUEST SPECIFIES
ONE OR MORE OF THE FOLLOWING ITEMS TO BE LISTED – ITEMS, AUTHORS,
```

DESCRIPTORS, TYPES, LANGUAGES, NAMES, OR TYPE IS COMMAND, ENTRY,
LIBRARY, OR PUBLIC.
OK..

describe then find
   DESCRIBE NAME IS 'NAME', I(1) . . . I(N)
   THE DESCRIBE REQUEST IN USED TO OBTAIN THE DOCUMENTATION OF A
PROGRAM WHOSE NAME IS KNOWN. THE INPUT OF THIS REQUEST GIVES
THE PROGRAM NAME AND THE NAMES OF THE DESIRED ITEMS OF INFORMATION.
IF NO ITEM NAME IS SPECIFIED THE INFORMATION ON ALL ITEMS WILL
BE PRINTED. FOR EXAMPLE TYPE –
   DESCRIBE DATE AND PURPOSE OF PROGRAM WHOSE NAME IS INFO
OK..

FIND I(1) IS V(1), . . . , I(N) IS V(N)
THE FIND REQUEST IS USED TO PERFORM A SEARCH FOR THE PROGRAM
OR PROGRAMS WHICH HAVE PARTICULAR INFORMATION ITEM VALUES.
THE ITEMS TO BE MATCHED ARE GIVEN BY TYPING THE ITEM NAMES TOGETHER
WITH THEIR ITEM VALUES. ACCEPTABLE ITEMS FOR SEARCHING ON ARE
– TYPE, LANGUAGE, DESCRIPTORS, AUTHOR, AND DATE. A DATE VALUE
MUST BE GIVEN IN THE FORM – DATE IS MM/DD/YY, WHERE MM IS A NUMERICAL
MONTH, DD IS A NUMERICAL DAY AND YY IS A NUMERICAL YEAR. ALL PROGRAMS
WHOSE DATE IS GREATER THAN THAT GIVEN WILL BE PRINTED. (I.E. THE MOST
RECENTLY DOCUMENTED PROGRAMS). DESCRIPTORS ARE SINGLE WORDS TYPED
IN ANY ORDER AND SEPARATED BY SPACES. A LIST OF DESCRIPTORS MAY BE
OBTAINED BY TYPING – LIST, DESCRIPTORS.
   FOR EXAMPLE TO FIND THE COMMANDS WHICH WERE DOCUMENTATED
SINCE JULY 4TH. AND HAVE AT LEAST THE DESCRIPTORS UTILITY TYPE –
FIND, DATE IS 6/04/64, DESCRIPTORS ARE DISK READ
OK..

store then edit then alter then remove
   STORE NAME IS 'NAME', FILE IS 'FILE', I(1) IS V(1), . . . , I(N) IS V(N)
   THE STORE REQUEST ENABLES ONE TO ENTER INFORMATION ABOUT A NEW
PROGRAM INTO AN INFORMATION FILE. THIS REQUEST REQUIRES INFORMATION
VALUES FOR EACH REQUIRED ITEM IN THE FORM – 'ITEM NAME' IS/ARE
'ITEM VALUE'. THE NAME OF NEW ITEMS MAY BE DEFINED BY TYPING THE NEW
ITEM NAME AND ITS VALUE. WHEN AN ITEM NAME IS PRINTED, TYPE THE
ITEM VALUE. ITEM NAMES AND ITEM VALUES OF OTHER ITEMS MAY BE
SUPPLIED BY TYPING A COMMA AFTER EACH ITEM VALUE THUS ANTICIPATING
THE REQUIRED INPUT AND REDUCING ON-LINE INTERACTION.
OK..

   EDIT NAME IS 'NAME'
   THE EDIT REQUEST CREATES A BCD FILE (AS A LINE MARKED FILE) FOR
USE IN MAKING CHANGES TO INFORMATION ITEMS USING SOME CTSS EDIT
PROCEDURE. THE FILE CREATED CONTAINS ALL INFORMATION ITEMS
EXCEPT THOSE ITEMS WHICH CAN BE USED WITH THE FIND REQUEST.
EACH INFORMATION ITEM IS PRECEDED BY A LINE GIVING THE ITEM
NAME PREFIXED BY A PERIOD (E.G.– .USAGE) AND CONSEQUENTLY NO
LINE OF AN ITEM SHOULD BEGIN WITH A PERIOD. THE FIRST NAME
OF THE FILE CREATED IS THE SAME AS THE PROGRAM NAME AND THE
SECOND IS 'INFO'.
OK..

   ALTER NAME IS 'NAME', I(1) IS V(1), . . . , I(N) IS V(N)
   THE ALTER REQUEST ALLOWS ONE TO CHANGE ITEM VALUES IN THE INFORMATION
DOCUMENTING A PROGRAM OR TO STORE ANOTHER INFORMATION ITEM.
THE ALTER REQUEST REQUIRES THAT THE PROGRAM NAME BE SPECIFIED.
OK..

   REMOVE NAME IS 'NAME', D IS 'D', A IS 'A', ITEM IS 'I'
   THE REMOVE REQUEST IS USED TO DELETE AN AUTHOR, DESCRIPTOR, OR
ITEM NAME FROM THE APPROPRIATE LIST OR TO REMOVE THE DOCUMENTATION
OF A PROGRAM FROM AN INFORMATION FILE. THIS REQUEST MIGHT BE
USED WHEN A PROGRAM IS DELETED FROM CTSS. VERIFICATION OF EACH
REQUEST TO REMOVE THE DOCUMENTATION OF A PROGRAM IS REQUIRED.
OK..

end
   5511 REGISTERS OF THE WORKSPACE WERE UNUSED.
R 49.300+34.216

APPENDIX B
LIST OF PUBLIC PROGRAMS

797 TRACKS USED ON 1/18/65

RUN	SAVED	–	DBGMEM	SAVED	–	L	SAVED	–
MONO4	SAVED	–	SQUALL	INFO	–	FILES	INFO	–
AEDLB1	BSS	–	DO	SAVED	–	DISTSS	SAVED	–
APPEND	SAVED	–	VARFIX	INFO	–	SUBUSE	SAVED	–
SQZBCD	BSS	–	(BLOOD)	BSS	–	VARFIX	SAVED	–
WHO	SAVED	–	SLEEP	SAVED	–	AEDLIB	BSS	–
DIS	SAVED	–	BLODI	INST	–	CTSS	SAVED	–
RELRW	SAVED	–	PADBCD	BSS	–	SAVFIL	SAVED	–
SAFE	SAVED	–	SAFE	INFO	–	WRFULL	BSS	–
PRSYMB	SAVED	–	PADBCD	SAVED	–	RERUN	SAVED	–
SQZBCD	SAVED	–	LISTCF	SAVED	–	LISTCF	INFO	–
FDOCT	SAVED	–	FDOCT	INFO	–	OPS2	WORDS	–
DSKLIB	BSS	–	SET	DATA	–	RUNBUG	SAVED	–
OPS2	BSS	–	OPS	SAVED	–	MACHI	BSS	–
STOMAP	SAVED	–	STOMAP	INFO	–	LIST	INFO	–
PLIST	SAVED	–	LIST	SAVED	–	QUES	SAVED	–
OCTPRT	SAVED	–	SUBLIS	DATA	–	CHAIN	DATA	–
LPREAD	DATA	–	DISX	SAVED	–	SLPLIB	BSS	–
MLIB	BSS	–	CMWRIT	DATA	–	CTEST	INFO	–
2AED	.LOAD	–	.RBIN	BSS	–	21	BSS	–
MAP	SAVED	–	BASIS	BSS	–	CONVT	BSS	–
TRANSF	BSS	–	INTEGR	BSS	–	DATA	SAVED	–
CONVOL	BSS	–	FIL	DIR	–	GETF	BSS	–
DIFFER	BSS	–	MINMAX	BSS	–	6T012	SAVED	–
GAME	BSS	–	ESLOPS	CRUNCH	–	TRACE	BSS	–
COMMND	LIST	–	APIAPT	SAVED	–	STR004	LINK	–
SUB	BSS	–	CTEST4	SAVED	–	STR002	LINK	–
STR003	LINK	–	STR005	LINK	–	STR006	LINK	–
OPL65	SAVED	–	STR007	LINK	–	BAYLES	BSS	–
CTEST3	INFO	–	UPDBSS	SAVED	–	DELRQ	SAVED	–
STR008	LINK	–	QUES	INFO	–	OLDRQ	SAVED	–
NEWCT3	SAVED	–	IF	INFO	–	IF	SAVED	–
SLAVE	SAVED	–		(MEMO)	–	COMAND	INFO	–
AEDBUG	BSS	–	AED002	SYSTEM	–	AED001	SYSTEM	–
STR001	LINK	–		SAVED	–	USER	REMARK	–

APPENDIX C
SUBROUTINE USAGE TABLE FOR TSLIB1

(This table was produced using the SUBUSE command written)
(by Barry Wolman. The table was produced on 12/22/64.)

ANA	IS NOT USED.					
ATN	IS NOT USED.					
BLK	IS CALLED BY ...	SEEK				
COM	IS NOT USED.					
COS	IS NOT USED.					
COT	IS NOT USED.					
DIM	IS NOT USED.					
EXP	IS CALLED BY ...	.01300				
FLK	IS CALLED BY ...	SEEK				
INT	IS NOT USED.					
LOG	IS CALLED BY ...	.01300				
MOD	IS NOT USED.					
ORA	IS NOT USED.					
SIN	IS NOT USED.					
SQR	IS NOT USED.					
TAN	IS NOT USED.					
ZEL	IS NOT USED.					
ACOS	IS NOT USED.					
ASIN	IS NOT USED.					
ATAN	IS NOT USED.					
BZEL	IS CALLED BY ...	(STB)	(RWT)			
COLT	IS CALLED BY ...	SCHAIN	BREAD	BWRITE	DREAD	
		SETVB	SEEK	XECOM	EOFXIT	
		MOVE1	DWRITE	SETERR		
DEAD	IS CALLED BY ...	EXIT				
DFAD	IS NOT USED.					
DFDP	IS CALLED BY ...	(IOH)				
DFMP	IS CALLED BY ...	(IOH)				
DFSB	IS NOT USED.					
DPNV	IS NOT USED.					
DUMP	IS NOT USED.					
ENDF	IS CALLED BY ...	BREAD	DREAD	SETVB	SEEK	
		EOFXIT	(RWT)			
EXIT	IS CALLED BY ...	(FPT)	.SET	BREAD	BWRITE	
		EOFXIT	RECOUP	FREE	LDUMP	
		SETVB				
FILE	IS CALLED BY ...	(RWT)				
FREE	IS NOT USED.					
FRET	IS NOT USED.					
GCLC	IS NOT USED.					
GCLS	IS NOT USED.					
GLOC	IS NOT USED.					
GMEM	IS NOT USED.					
GNAM	IS CALLED BY ...	SCHAIN	BREAD	BWRITE	DREAD	
		SETVB	SEEK	FSTAT	RENAME	
		CHMODE	DELETE	COMARG	COMFL	
		PRNTP	RDFLXB	GMEM	CHNCOM	
		DWRITE	DSKDMP	GCLC	GETCF	
INDV	IS NOT USED.					
.BSF	IS NOT USED.					
.BSR	IS NOT USED.					
.EFT	IS NOT USED.					
.MTX	IS NOT USED.					
.RWT	IS NOT USED.					
.SET	IS NOT USED.					

```
MAXO    IS NOT USED.
MAX1    IS NOT USED.
MINO    IS NOT USED.
MIN1    IS NOT USED.
SCLC    IS NOT USED.
SCLS    IS NOT USED.
SEEK    IS CALLED BY ... (STB)      (TSH)      SCHAIN
SETU    IS NOT USED.
SFDP    IS NOT USED.
SIGN    IS NOT USED.
SLOC    IS NOT USED.
SMEM    IS NOT USED.
SNAP    IS CALLED BY ... (SCH)      (TSH)      BREAD      BWRITE
                         DWRITE     SETVB      SEEK       DSKDMP
                         (RWT)      SYPAR      SETERR     DREAD
                         DELETE
SQRT    IS CALLED BY ... .01300
SRCH    IS CALLED BY ... SEEK       (RWT)
TANH    IS NOT USED.
XDIM    IS NOT USED.
XFIX    IS NOT USED.
XINT    IS NOT USED.
XLOC    IS NOT USED.
XMOD    IS NOT USED.

BCDEC   IS CALLED BY ... EOFXIT
BREAD   IS CALLED BY ... (STB)
CLOUT   IS CALLED BY ... EXIT
COMFL   IS NOT USED.
DEFBC   IS CALLED BY ... GTNAM      (RWT)
DELBC   IS CALLED BY ... (STB)
DERBC   IS NOT USED.
DREAD   IS CALLED BY ... SCHAIN
ENDRD   IS CALLED BY ... (TSH)
ERASE   IS NOT USED.
ERROR   IS CALLED BY ... (FPT)      .01300     EXP(2      ACOS
                         LOG        SQRT       TAN        INDV
                         EXP
EXITM   IS NOT USED.
EXMEM   IS NOT USED.
EXP(1   IS NOT USED.
EXP(2   IS NOT USED.
EXP(3   IS NOT USED.
FSTAT   IS CALLED BY ... (STB)      GTNAM
GETCF   IS NOT USED.
GETTM   IS NOT USED.
GTNAM   IS CALLED BY ... SCHAIN     XECOM
IOEND   IS NOT USED.
IOITR   IS NOT USED.
IOPAR   IS NOT USED.
IOSCP   IS NOT USED.
IOSET   IS NOT USED.
.DUMP   IS CALLED BY ... DSKDMP
.FILE   IS CALLED BY ... SEEK       SRCH       (RWT)
.LOAD   IS CALLED BY ... DSKDMP     SYPAR
.LOOK   IS NOT USED.
.READ   IS NOT USED.
.SAVE   IS NOT USED.
.SEEK   IS CALLED BY ... SEEK
JOBTM   IS NOT USED.
LDUMP   IS CALLED BY ... .01300     EXP(2      ACOS       EXP
                         SQRT       TAN        INDV       LOG
LJUST   IS CALLED BY ... DELETE
MOVE1   IS CALLED BY ... SCHAIN     XECOM      GCLC
MOVE2   IS CALLED BY ... SCHAIN     XECOM      GCLC
MOVE3   IS CALLED BY ... SCHAIN     XECOM      GCLC
OCABC   IS CALLED BY ... (FPT)
```

OCDBC	IS CALLED BY ...	SNAP			
OCLBC	IS NOT USED.				
OCRBC	IS NOT USED.				
PDUMP	IS NOT USED.				
PRNTP	IS NOT USED.				
RANNO	IS NOT USED.				
RDFLX	IS CALLED BY ...	.RDATA	(CSH)	DELETE	
RJUST	IS CALLED BY ...	SCHAIN	XECOM		
SETVB	IS NOT USED.				
SLEEP	IS NOT USED.				
STQUO	IS NOT USED.				
SYPAR	IS NOT USED.				
TIMER	IS NOT USED.				
VREAD	IS NOT USED.				
WRFLX	IS CALLED BY ...	STOMAP	(FPT)	.PCOMT	.RDATA
		ERROR	SCHAIN	BREAD	BWRITE
		SETVB	SEEK	(IOH)	(EXE)
		EOFXIT	SETERR	SNAP	FREE
		PRNTP	(SPH)	DREAD	(RWT)
		EXIT			
XECOM	IS NOT USED.				
XMAXO	IS NOT USED.				
XMAX1	IS NOT USED.				
XMINO	IS NOT USED.				
XMIN1	IS NOT USED.				
XSIGN	IS NOT USED.				
(BST)	IS CALLED BY ...	.BSF			
(CSH)	IS NOT USED.				
(EFT)	IS CALLED BY ...	.BSF			
(EXE)	IS CALLED BY ...	(IOH)			
(FIL)	IS CALLED BY ...	DWRITE			
(FPT)	IS CALLED BY ...	.SETUP			
(IOH)	IS CALLED BY ...	(SCH)	(SPH)	(CSH)	(TSH)
		DWRITE	DREAD		
(RLR)	IS NOT USED.				
(RTN)	IS CALLED BY ...	DREAD			
(RWT)	IS CALLED BY ...	.BSF			
(SCH)	IS NOT USED.				
(SLI)	IS NOT USED.				
(SLO)	IS NOT USED.				
(SPH)	IS NOT USED.				
(STB)	IS NOT USED.				
(STH)	IS NOT USED.				
(TSB)	IS NOT USED.				
(TSH)	IS NOT USED.				
(WLR)	IS NOT USED.				
AKNOLG	IS NOT USED.				
APPEND	IS CALLED BY ...	(SCH)	(STB)		
ASSIGN	IS CALLED BY ...	(SCH)	(STB)		
BWRITE	IS CALLED BY ...	(STB)			
CHMODE	IS NOT USED.				
CHNCOM	IS CALLED BY ...	SCHAIN	EXIT		
CLKOUT	IS NOT USED.				
CLOCOF	IS NOT USED.				
CLOCON	IS NOT USED.				
COMARG	IS NOT USED.				
COMFIL	IS CALLED BY ...	COMFL			
DCEXIT	IS NOT USED.				
DELETE	IS CALLED BY ...	SCHAIN	XECOM	GTNAM	RENAME
DORMNT	IS CALLED BY ...	EXIT			
DSKDMP	IS NOT USED.				
DSKLOD	IS NOT USED.				
DWRITE	IS NOT USED.				
ENDJOB	IS CALLED BY ...	.PRSLT	DFAD		
EOFXIT	IS CALLED BY ...	(TSH)	BREAD	DREAD	SETVB

```
FWRITE    IS NOT USED.
GETBRK    IS NOT USED.
GETCFN    IS NOT USED.
GETCLC    IS CALLED BY ...  SCHAIN    XECOM     GCLC
GETCLS    IS CALLED BY ...  SCHAIN    XECOM     GCLC
GETCOM    IS CALLED BY ...  COMARG
GETIME    IS NOT USED.
GETLOC    IS CALLED BY ...  SYPAR
GETMEM    IS CALLED BY ...  BREAD     BWRITE    SRCH      FREE
                            GMEM
IOHSIZ    IS NOT USED.
.01300    IS NOT USED.
.01301    IS NOT USED.
.01311    IS NOT USED.
.03310    IS NOT USED.
.03311    IS CALLED BY ...  .RDATA    (IOH)
.APEND    IS CALLED BY ...  SEEK
.ASIGN    IS CALLED BY ...  SEEK
.COMNT    IS NOT USED.
.DLETE    IS CALLED BY ...  SCHAIN    XECOM     DELETE
.ENDRD    IS CALLED BY ...  SEEK      SRCH      (RWT)
.ERASE    IS CALLED BY ...  DELETE
.FILDR    IS NOT USED.
.FSTAT    IS CALLED BY ...  (SCH)     SCHAIN    FSTAT     RENAME
                            DELETE
.PCOMT    IS NOT USED.
.PNCHL    IS NOT USED.
.PRBCD    IS NOT USED.
.PRINT    IS CALLED BY ...  .PRSLT
.PROCT    IS NOT USED.
.PRSLT    IS NOT USED.
.PUNCH    IS NOT USED.
.RDATA    IS NOT USED.
.READK    IS CALLED BY ...  (TSH)     BREAD     DREAD     SETVB
.READL    IS NOT USED.
.RELRW    IS NOT USED.
.RENAM    IS CALLED BY ...  RENAME    CHMODE    DELETE
.RESET    IS NOT USED.
.RPDTA    IS NOT USED.
.RSTOR    IS NOT USED.
.RSTRN    IS NOT USED.
.SAVRN    IS NOT USED.
.SCRDS    IS NOT USED.
.SETUP    IS NOT USED.
.SPRNT    IS CALLED BY ...  .PRSLT     .SET
.TAPRD    IS NOT USED.
.TAPWR    IS NOT USED.
.WRITE    IS CALLED BY ...  (SCH)     BWRITE    DWRITE    SETVB
KILLTR    IS NOT USED.
MOVIE)    IS CALLED BY ...  STOMAP    (FPT)
PRNTPA    IS NOT USED.
PRNTPC    IS NOT USED.
RDFLXA    IS CALLED BY ...  RDFLXB
RDFLXB    IS NOT USED.
RDFLXC    IS NOT USED.
RECOUP    IS CALLED BY ...  SEEK      (EXE)     SNAP
RENAME    IS NOT USED.
RSCLCK    IS NOT USED.
RSTRTN    IS NOT USED.
SAVBRK    IS NOT USED.
SCHAIN    IS NOT USED.
SETBCD    IS NOT USED.
SETBRK    IS NOT USED.
SETCLC    IS CALLED BY ...  SCHAIN    XECOM     GCLC
SETCLS    IS CALLED BY ...  SCHAIN    XECOM     GCLC
SETEOF    IS CALLED BY ...  SCHAIN
```

```
SETERR    IS NOT USED.
SETFMT    IS NOT USED.
SETFUL    IS NOT USED.
SETLOC    IS NOT USED.
SETMEM    IS CALLED BY ...  BREAD     BWRITE     SRCH       FREE
                            GMEM
SETNAM    IS NOT USED.
SETVBF    IS NOT USED.
STOMAP    IS NOT USED.
STOPCL    IS NOT USED.
TIMLFT    IS NOT USED.
TSSFIL    IS CALLED BY ...  SYPAR
USRFIL    IS CALLED BY ...  SYPAR
VWRITE    IS NOT USED.
WRDCNT    IS NOT USED.
WRFLXA    IS CALLED BY ...  DELETE    PRNTP
XDETRM    IS NOT USED.
XSIMEQ    IS NOT USED.
(EFTM)    IS CALLED BY ...  .SETUP    .RDATA     (IOH)
(LFTM)    IS CALLED BY ...  (FPT)     .RDATA     (IOH)
(SPHM)    IS NOT USED.
(STHM)    IS NOT USED.
(TSHM)    IS NOT USED.

MISSING SUBROUTINES...

MOVIE)
```

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Lincoln Laboratory, M.I.T.	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP None

3. REPORT TITLE

On-Line Documentation of the Compatible Time-Sharing System

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Technical Report

5. AUTHOR(S) *(Last name, first name, initial)*

Winett, Joel M.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
12 May 1965	52	7

8a. CONTRACT OR GRANT NO. AF 19(628)-500 b. PROJECT NO. 649L c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) TR-387 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* ESD-TDR-65-68

10. AVAILABILITY/LIMITATION NOTICES

None

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Air Force Systems Command, USAF

13. ABSTRACT

The dissemination of information about computer programs is hampered because of the lack of conformity in documentation, the delays inherent in any distribution system, and the inability to select only desired information without being flooded with information which is not of present interest. An on-line system for storing and retrieving information about the programs associated with the Compatible Time-Sharing System (CTSS) has been developed to be included as a CTSS command. This system will help to document the system commands, supervisor entries, library subprograms, and public programs. These types of programs have been chosen since there is an urgent need for having this documentation available on demand, i.e., on-line.

14. KEY WORDS

documentation
computer programs
catalogs
information retrieval
information systems
computer applications
on-line systems
time sharing
data storage